2007-03

An evaluation methodology for protocol analysis systems

Hoffmeister, Chris W.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

AN EVALUATION METHODOLOGY FOR PROTOCOL ANALYSIS SYSTEMS

by

Chris W. Hoffmeister

March 2007

Thesis Advisor:	George W. Dinolt
Thesis Co-Advisor:	Jonathan Herzog

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704-0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY	**2. REPORT DATE** March 2007	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE: An Evaluation Methodology for Protocol Analysis Systems	**5. FUNDING NUMBERS**
6. AUTHOR(S) Chris W. Hoffmeister	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(maximum 200 words)***

Current day communication systems rely on protocols to provide secure communications among parties. Weaknesses in protocols, at first thought to be secure, have been found through deep analysis. There are many systems that have been designed to provide a means to test the various security characteristics of communication protocols. We present an evaluation methodology that can be used to evaluate protocol analysis systems based on their scope, correctness, performance, and usability characteristics. We apply portions of the methodology to a set of protocol analysis systems to show the evaluation methodology in action.

14. SUBJECT TERMS Evaluation Methodology, Protocol Analysis, Cryptographic Protocol Shapes Analyzer (CPSA), ProVerif	**15. NUMBER OF PAGES** 263
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	**18. SECURITY CLASSIFICATION OF THIS PAGE**	**19. SECURITY CLASSIFICATION OF ABSTRACT**	**20. LIMITATION OF ABSTRACT**
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500 Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18-298-102

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited.

AN EVALUATION METHODOLOGY FOR PROTOCOL ANALYSIS SYSTEMS

Chris W. Hoffmeister
Lieutenant, United States Navy
B.S., Texas A&M University, 2001

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
MARCH 2007

Author: Chris W. Hoffmeister

Approved by: George W. Dinolt
Thesis Advisor

Jonathan Herzog
Thesis Co-Advisor

Peter Denning
Chairman, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Current day communication systems rely on protocols to provide secure communications among parties. Weaknesses in protocols, at first thought to be secure, have been found through deep analysis. There are many systems that have been designed to provide a means to test the various security characteristics of communication protocols. We present an evaluation methodology that can be used to evaluate protocol analysis systems based on their scope, correctness, performance, and usability characteristics. We apply portions of the methodology to a set of protocol analysis systems to show the evaluation methodology in action.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

I. INTRODUCTION 1

II. BACKGROUND 5

A. PROTOCOL ANALYSIS FIELD 5

B. EVALUATION EFFORTS TO DATE 7

C. THE NEED FOR AN EVALUATION METHODOLOGY 7

III. EVALUATION METHODOLOGY 9

A. CRITERIA 9

1. Scope 10

2. Correctness 13

3. Performance 16

4. Usability 17

B. METHODS 20

1. Pre-Testing 21

2. Scope Testing 21

3. Correctness Testing 22

4. Performance Testing 24

5. Usability Testing 25

6. Post-Testing 27

7. Test Ordering 28

C. RATINGS AND WEIGHTS 30

1. Scope 32

2. Correctness 34

3. Performance 36

4. Usability 39

5. Overall Weightings 40

D. METHODOLOGY ADAPTATIONS 45

1. Criteria 46

2. Methods 47

3. Ratings and Weights 49

IV. EXAMPLE EVALUATIONS 51
A. CRYPTOGRAPHIC PROTOCOL SHAPES ANALYZER 51
1. Scope 52
2. Correctness 54
3. Performance 55
4. Usability 56
5. Overall 56
B. PROVERIF 56
1. Scope 57
2. Correctness 59
3. Performance 60
4. Usability 61
5. Overall 61
V. CONCLUSIONS 63
A. FUTURE WORK 63
APPENDIX A: TEST PLATFORM CONFIGURATION 65
APPENDIX B: FILE LISTINGS 67
APPENDIX C: EVALUATION DATA 105
APPENDIX D: LIST OF PROTOCOL ANALYSIS SYSTEMS 235
LIST OF REFERENCES 237
FURTHER READING 241
INITIAL DISTRIBUTION LIST 243

LIST OF FIGURES

Figure 1: High Level Functional View of the Evaluation Methodology 4
Figure 2: Needham-Schroeder Public-Key Protocol in Standard Protocol Notation..... 6
Figure 3: Attack on Needham-Schroeder Public-Key Protocol in Standard Protocol Notation..... 6
Figure 4: Required Testing Order 29
Figure 5: Recommended Testing Order 30
Figure 6: System Execution Time CI^{+} Calculation 36

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF LISTINGS

Listing 1: Master Test Script 69
Listing 2: CPSA Batch Mode NS Simple 2 Protocol Specification 71
Listing 3: CPSA Batch Mode NS Full 2 Protocol Specification 73
Listing 4: CPSA Batch Mode NSL Simple 2 Protocol Specification 75
Listing 5: CPSA Batch Mode NSL Full 2 Protocol Specification 77
Listing 6: CPSA Batch Mode Kerberos Protocol Specification 80
Listing 7: CPSA Batch Mode NS Simple 2 Protocol Test Script 81
Listing 8: CPSA Batch Mode NS Full 2 Protocol Test Script 82
Listing 9: CPSA Batch Mode NSL Simple 2 Protocol Test Script 83
Listing 10: CPSA Batch Mode NSL Full 2 Protocol Test Script 85
Listing 11: CPSA Secondary Memory Requirement Test Script 85
Listing 12: ProVerif Spi Calculus NS Full 2 Protocol Specification 90
Listing 13: ProVerif Spi Calculus NSL Full 2 Protocol Specification 94
Listing 14: ProVerif Spi Calculus Kerberos Protocol Specification 99
Listing 15: ProVerif Spi Calculus NS Full 2 Protocol Test Script 100
Listing 16: ProVerif Spi Calculus NSL Full 2 Protocol Test Script 101
Listing 17: ProVerif Spi Calculus Kerberos Protocol Test Script 102
Listing 18: ProVerif Secondary Memory Requirement Test Script 103
Listing 19: CPSA NS Simple 2 Observed Correctness Test Data 109
Listing 20: CPSA NS Full 2 Observed Correctness Test Data 113
Listing 21: CPSA NSL Simple 2 Observed Correctness Test Data 117
Listing 22: CPSA NSL Full 2 Observed Correctness Test Data 121
Listing 23: CPSA Secondary Memory Requirement Criterion Test Data 124
Listing 24: ProVerif NS Full 2 Observed Correctness Test Data 172
Listing 25: ProVerif NSL Full 2 Observed Correctness Test Data 198
Listing 26: ProVerif Kerberos Observed Correctness Test Data 228
Listing 27: ProVerif Secondary Memory Requirement Criterion Test Data 232

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF TABLES

Table 1: Protocol Analysis System Results x Protocol Possibilities 14
Table 2: Criterion Rating 31
Table 3: System Type Support Criterion Rating 32
Table 4: System Operation Support Criterion Rating 33
Table 5: System Session Type Criterion Rating 33
Table 6: System Theoretical Testable Protocol Characteristics Criterion Rating 33
Table 7: System Correctness Criterion Rating 34
Table 8: System Execution Time Criterion Rating 37
Table 9: System Secondary Memory Requirement Criterion Rating 38
Table 10: System Main Memory Requirement Criterion Rating 38
Table 11: System Automation Criterion Rating 39
Table 12: System Specification Comments Criterion Rating 39
Table 13: Scope Criteria Overall Weightings 43
Table 14: Correctness Criteria Overall Weightings 44
Table 15: Performance Criteria Overall Weightings 45
Table 16: Objective Usability Criteria Overall Weightings 45
Table 17: CPSA Type Support Criterion Results 52
Table 18: CPSA Operation Support Criterion Results 53
Table 19: CPSA Theoretical Testable Protocol Characteristics Criterion Results 54
Table 20: CPSA Theoretical Correctness Criterion Results 54
Table 21: CPSA Observed Correctness Criterion Results 55
Table 22: CPSA Execution Time Criterion Results 55
Table 23: CPSA Main Memory Requirement Criterion Results 56
Table 24: ProVerif Type Support Criterion Results 57
Table 25: ProVerif Operation Support Criterion Results 58
Table 26: ProVerif Theoretical Testable Protocol Characteristics Criterion Results 59
Table 27: ProVerif Theoretical Correctness Criterion Results 59
Table 28: ProVerif Observed Correctness Criterion Results 60
Table 29: ProVerif Execution Time Criterion Results 60

Table 30: ProVerif Main Memory Requirement Criterion Results 61
Table 31: CPSA NS Full 2 Execution Criterion Test Data 122
Table 32: CPSA NSL Full 2 Execution Criterion Test Data 123
Table 33: CPSA NS Full 2 Main Memory Requirement Criterion Test Data 125
Table 34: CPSA NSL Full 2 Main Memory Requirement Criterion Test Data 126
Table 35: ProVerif NS Full 2 Execution Time Criterion Test Data 229
Table 36: ProVerif NSL Full 2 Execution Time Criterion Test Data 230
Table 37: ProVerif Kerberos Execution Time Criterion Test Data 231
Table 38: ProVerif NS Full 2 Main Memory Requirement Criterion Test Data 232
Table 39: ProVerif NSL Full 2 Main Memory Requirement Criterion Test Data 233
Table 40: ProVerif Kerberos Main Memory Requirement Criterion Test Data 234
Table 41: List of Protocol Analysis Systems 236

LIST OF SYMBOLS, ACRONYMS AND ABBREVIATIONS

CPSA:	Cryptographic Protocol Shapes Analyzer
DIMM:	Dual In-line Memory Module
GB:	Giga Byte
GHz:	Giga Hertz
HoQ:	House of Quality
IETF:	Internet Engineering Task Force
MB:	Megabyte
MMR:	Main Memory Requirement
NS:	Needham-Schroeder
NSL:	Needham-Schroeder-Lowe
OCaml:	Objective Caml
POSIX:	Portable Operating System Interface for uniX
QFD:	Quality Function Deployment
RAM:	Random Access Memory
RFC:	Request For Comments
SMR:	Secondary Memory Requirement

THIS PAGE INTENTIONALLY LEFT BLANK

ACKNOWLEDGEMENT

I want to thank Professor Dinolt and Professor Herzog for their guidance and patience during the work in performing this investigation. The work presented here would not have been possible were it not for the many discussions we had and the direction they provided. I want to thank Professor Sadagic for getting us on the right path in regards to the usability aspect of this project.

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

In networked communications there are numerous protocols that are layered on top of each other to provide confidential, reliable, and trusted communications across insecure and unreliable communication paths. There are numerous communication protocols that are used for various forms of communication. Some protocols do not make any claims or attempts to provide confidential, reliable, or trusted communications; other protocols make significant claims and attempt to provide some or all of these properties. As in other subject areas, an object's ability to meet the specified requirements should not be taken "as is" but instead must be tested and analyzed; communication protocols are no different. There are examples of communication protocols that were thought to be secure but later found to be flawed. Protocol Analysis is the field of study primarily concerned with the analyzing and designing communication protocols and proving the security claims of the protocols.

Protocol Analysis has been a field of interest for some time and has become an ever important field of study in today's world. Numerous protocol analysis systems exist to aid the designer and evaluator in their efforts, such as the Cryptographic Protocol Shapes Analyzer (CPSA) [GUTT2006], ProVerif [BLAN2001], Spin [HOLZ1994], and SPEAR II [SAUL1999, SAUL2001b, SAUL2003]. Some analysis systems excel in analyzing a protocol's ability to perform across a noisy channel, such as Spin; others are used to prove confidentiality and authenticity characteristics, such as CPSA, ProVerif, and SPEAR II. In this project we are interested in communication protocol analysis systems that can be used to prove confidentiality and authenticity characteristics of communication protocols. These types of protocols are commonly referred to as security protocols[1].

Just as communication protocols are analyzed and evaluated, so too must the protocol analyzers be analyzed and evaluated. Protocol analysis system evaluation efforts to date have mainly been focused on performance criteria [BASI2003, HOPP2000,

[1] In this work we use the term protocol to mean security protocol unless otherwise noted.

LOWE1997] and to some degree usability [SAUL2001b]. We are not aware of any overarching evaluation methodology tailored to protocol analysis systems.

The goal of this project was to devise an evaluation methodology that can be used to compare different protocol analysis systems. In this project we present the first evaluation methodology that can be used to evaluate and analyze protocol analysis systems. The evaluation methodology consists of criteria, methods, and weights. The criteria are the topics that the analysis systems are evaluated on. The methods are the testing mechanisms for the criteria. The weights include two things: the rating system for each criterion, and the weightings used between criteria. Figure 1 is a high level functional view of the evaluation methodology presented in this project. At the top of Figure 1 is the set of protocol analysis systems that are to be evaluated by the evaluation methodology user. The user evaluates a single protocol analysis system at a time. The user evaluates the analysis systems on four categories of criteria: scope, correctness, performance, and usability. The user inputs the results from the testing methods into the overall weighting function. The output from the weighting function is the ultimate output of the evaluation methodology.

The results from the evaluation methodology can be used in a number of ways. First, protocol analysis system designers can use the evaluation results to analyze how different versions of their analysis system meet the needs of the end users. Secondly, evaluation results from separate protocol analysis systems can be compared and further analyzed. The comparative analysis of protocol systems can be used by evaluators to determine which protocol analysis system best meets their needs.

One of the key aspects of the evaluation methodology we present is the ability of the methodology to address the needs of the evaluator through the weighting function. Different evaluators will use protocol analysis systems for different purposes and will thus have different needs that the analysis systems should meet. For example, an evaluator may be concerned with the ease at which students are able to specify protocols in the analysis system, or an evaluator may be concerned with the amount of resources an analysis system uses. The factors in the weighting function can be adapted to address the needs of a wide range of evaluators. The weighting factors we present are an example

based on the needs of a communications system manager, such as an Information Systems Security Manager (ISSM) [CNSS4009]. One of the responsibilities of a communications system manager is to decide which protocols will be implemented on the communications network. This responsibility should not be taken lightly by any manager and is growing in size because of the increasing number of protocols that are introduced to and published by the Internet Engineering Task Force (IETF). A manager needs a protocol analysis system that can aid him in the decision making process. Analysis systems that do not aid the manager in making a decision among various communication protocols do not present the highest degree of usefulness to the communications system manager. Our example weights award more points to analysis systems that aid in the decision making process over analysis systems that do not aid in the decision making process.

Another key aspect of the methodology we present is related to the "living" nature that we expect the methodology to have. In addition to the evaluation methodology itself, we present ways by which the methodology can be adapted to meet the needs of the growing protocol analysis field. We present informal reasons as to why the methodology would need to be adapted and processes for making sound adaptations. We additionally discuss the possibilities of comparing data collected under one version of the evaluation methodology with a different version. The modular manner in which we present the methodology, in part, allows the methodology to be adapted.

In summary, the key result of this project is that we show that protocol analysis systems can be evaluated against the needs of an evaluator. The results returned by this evaluation methodology can be used for various purposes by different parties, and the methodology can be tailored to address the needs of a wide range of evaluators. Additionally, the mechanisms we present to adapt the evaluation methodology give the methodology the ability to be a living methodology that grows with the protocol analysis field.

In Chapter II we discuss background information pertaining to the protocol analysis field and general, and information specific to the evaluation methodology we present. In Chapter III we present the details of the evaluation methodology. In Chapter

IV we show the validity of the evaluation methodology by applying part of the methodology to a set of protocol analysis systems. Finally, we conclude and discuss future work in Chapter V.

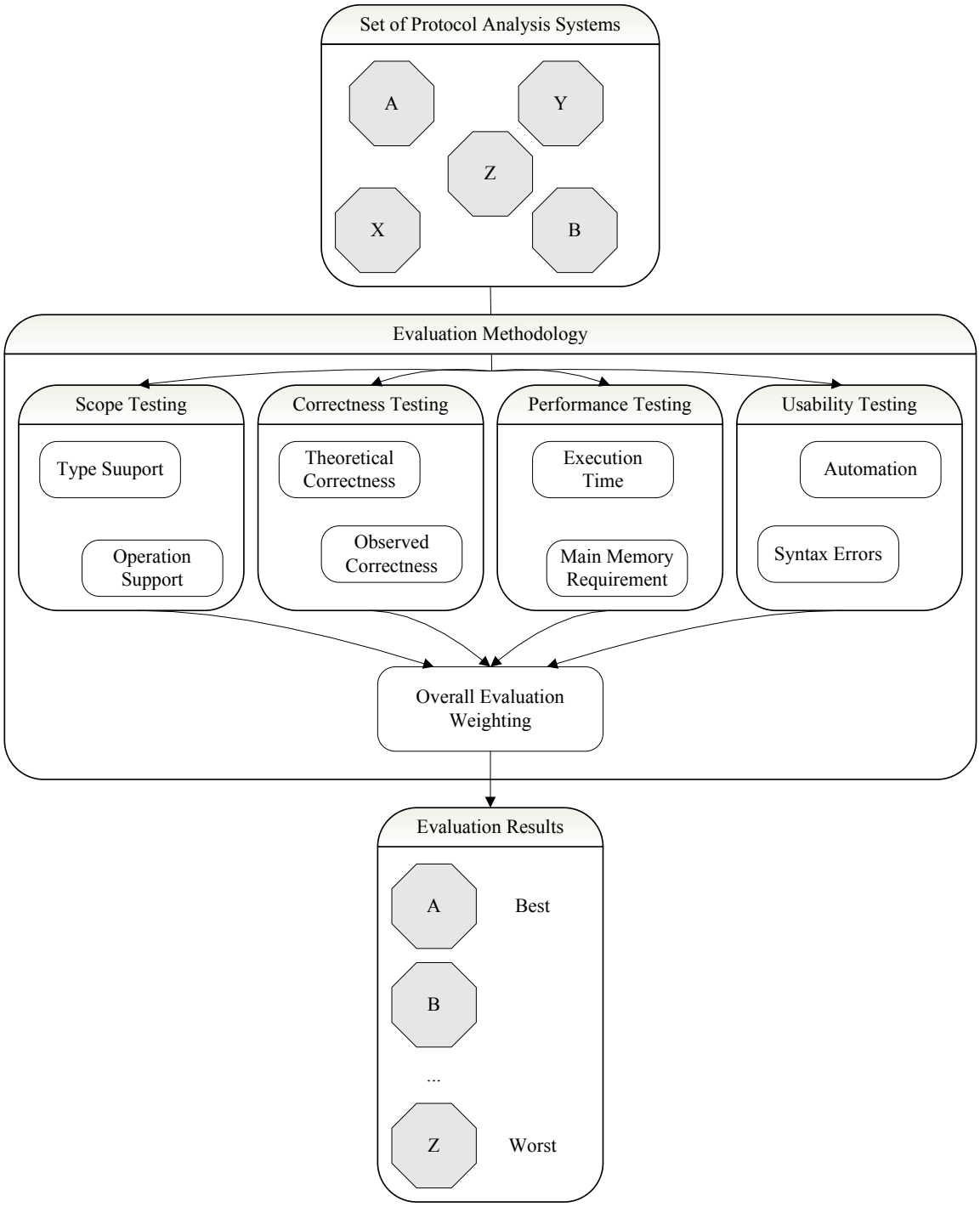

Figure 1: High Level Functional View of the Evaluation Methodology

II. BACKGROUND

In this section we present requisite background information. We begin with an introduction to the protocol analysis field. We then discuss the evaluation efforts that have occurred to date. We end with a discussion as to why an evaluation methodology is needed.

A. PROTOCOL ANALYSIS FIELD

As with any field, there is a specialized language used within the protocol analysis community. Figure 2 serves as an introduction to the terminology used in the protocol analysis field; specifically Figure 2 is the simplified representation of the Needham-Schroeder (NS) public-key encryption protocol [NEED1978].

A protocol consists of a series of transactions between the communicating parties; each transaction or step of the protocol is designated on a separate line as in the example. A step of a protocol is comprised of two components, the first being the communicating parties and the second the formatted message contents. A colon (:) is used to separate the communicating parties from the formatted message contents. An arrow ($\rightarrow$) is used to separate the sender of the message from the receiver. For example, the first step in Figure 2 states that *A* is sending the message to *B*. Within the community Alice (A) typically initiates the communications with Bob (B) the responder, third party servers (S) are also used within in some protocols. When an attack is found in a protocol, the attack is shown in the same format with infiltrator (I) inserted into the message listings, as exemplified in Figure 3; in literature the attacker or infiltrator is commonly named Eve, due to the act of eavesdropping.

Messages can consist of a variety of terms, such as the names of the participating parties or principals, nonces (N), timestamps (T), keys (K), text, numbers, or other variables. There are operations that can be performed on message terms, the three most common are concatenation, encryption, and hashing. The concatenation of terms is typically denoted with a comma (,) between the concatenated terms. The encryption of terms is noted by enclosing the terms to be encrypted in brackets followed by the key used to perform the encryption. For example, the message in the first line of Figure 2,

$\{A, N_A\} K_B$, means that the terms A and N_A are concatenated and then encrypted with the key K_B. Hashing is typically noted in a similar fashion as other mathematical functions like sine and cosine, where the inputs to the hash function are enclosed in parentheses; for example, $A \rightarrow B: \{A, N_A\} K_B, \text{hash}(A, N_A)$. Other mathematical functions such as addition and xor also appear in protocols and are represented by the standard conventions, a plus sign (+) for addition and a plus sign within a circle ($\oplus$) for xor.

$$A \rightarrow B: \{A, N_A\} K_B$$
$$B \rightarrow A: \{N_A, N_B\} K_A$$
$$A \rightarrow B: \{N_B\} K_B$$

Figure 2: Needham-Schroeder Public-Key Protocol in Standard Protocol Notation

$$A \rightarrow I: \{A, N_A\} K_I$$
$$I \rightarrow B: \{A, N_A\} K_B$$
$$B \rightarrow I: \{N_A, N_B\} K_A$$
$$I \rightarrow A: \{N_A, N_B\} K_A$$
$$A \rightarrow I: \{N_B\} K_B$$
$$I \rightarrow B: \{N_B\} K_B$$

Figure 3: Attack on Needham-Schroeder Public-Key Protocol in Standard Protocol Notation[2]

Over the years there have been many protocols developed, some provide security guaranties others do not. For instance the TCP/IP version 4 does not provide any security guaranties, nor does the NS protocol. There are protocols that do provide security guaranties, such as the Needham-Schroeder-Lowe (NSL) protocol [LOWE 1996]. The NS protocol was first shown to be secure through the use of BAN logic [BURR1990]; six years later and almost 20 years after the NS protocol was developed, Lowe found a flaw in the NS protocol using FDR [LOWE1996].

Just as there have been many protocols developed over the years there have been many theories for proving security protocols correct. Additionally, these theories have been incorporated into automated analysis systems such as CPSA and ProVerif. Today,

[2] The attack on and fix to the Needham-Schroeder public-key protocol was first published by Lowe in [LOWE1996].

there are many protocols, and many analysis systems to analyze the protocols. The question arises as to how the protocol analysis systems themselves can be compared and evaluated.

B. EVALUATION EFFORTS TO DATE

Evaluation efforts of protocol analysis systems to date have not been as comprehensive as the evaluation methodology we present. The most abundant form of protocol analysis system evaluation efforts have focused on performance criteria, such as execution time and secondary memory requirements [BASI2003, HOPP2000, LOWE1997]. There has also been an effort to evaluate the usability of a protocol analysis system [SAUL2001b]. The key issue to note about these analysis efforts is that they were conducted independently, i.e., there was not a common set of criteria or testing methods used among these efforts. The independent nature of these evaluation efforts does not lead to results that can be easily compared.

There have also been survey efforts that describe numerous protocol analysis systems [HEAL2004, LOPE2006, MEAD2003]. As with the previously mentioned evaluation efforts the various survey efforts have occurred independently of each other. Although these efforts have compared various protocol analysis systems in a single body of work, the survey nature of them does not lead to a clear determination of which analysis system would best meet the needs of system users.

We also found that in general the evaluation efforts to date have not addressed the different user classes that might apply protocol analysis systems. We see a number of different user classes for protocol analysis systems; such as communications system managers, communication protocol designers, and educators. These classes of users clearly have varying needs for protocol analysis systems. To our knowledge these varying needs have not been addressed to date.

C. THE NEED FOR AN EVALUATION METHODOLOGY

The evaluation efforts to date have failed to effectively compare and evaluate protocol analysis systems. The efforts have either been singular in nature that cannot be compared, or high-level summaries that are not detailed enough. In general, the

evaluation efforts to date have been ad-hoc. We address this by presenting a formal evaluation methodology that allows protocol analysis systems to be compared and evaluated in detail.

We see a need for the evaluation methodology we present in this research. There are a number of reasons why we see this need. First, over the years the protocol analysis field has grown and continues to see new analysis systems and theories evolve. The continuing increase in the number of analysis systems clearly raises the need to determine which analysis systems should be used. Second, the evaluation efforts to date have been independent in nature. As previously mentioned, the independent nature of these efforts does not allow for comparative analysis of the evaluation results. Third, the survey efforts to date have not provided detailed enough information to determine which analysis systems should be used and which should not. Fourth, the varying needs of protocol analysis system user classes have not been addressed. We feel this is partly due to the lack of a means to assess protocol analysis systems based on the needs of the system user.

We address the need for an evaluation methodology in this work. We present a methodology that is detailed, able to tailored, that produces comparable results, and is adaptable. The evaluation methodology provides greater detail than high level survey efforts. The methodology can be tailored to address the needs of the various analysis system user classes. The results from various runs of the methodology can be compared; allowing the evaluator to determine which analysis system best meets their needs. Lastly, the methodology we present can grow with the protocol analysis field and time.

III. EVALUATION METHODOLOGY

In this section we present the evaluation methodology and the core components that are present in an evaluation methodology. The components are presented in a modular fashion to show that they can be discussed independent of each other and to aid in modifications to adapt the methodology to meet an expanding set of needs. We begin by defining and discussing the *criteria* that are relevant in performing an evaluation of protocol analysis systems, as the other components of the methodology are dependent on the criteria. We then present the *methods* that are used to evaluate the protocol analysis systems on the relevant criteria. The last core components of any evaluation methodology are the *ratings* and *weights* that are used within each criterion and in between the criteria. After we define the components of our evaluation methodology we discuss ways in which the methodology can be adapted and how results across some adaptations can be compared.

A. CRITERIA

The core behind an evaluation methodology is the criteria that comprise it. In this research we present criteria that are grouped into four main categories: scope, correctness, performance, and usability. An underlying design principle of the criteria we present here is that each of the criteria provides an independent way in which to evaluate a protocol analysis system. At first glance one would expect that the correctness criteria would be discussed first as correctness is often the most important characteristic of a process. We do not present the correctness criteria first; instead we present the scope criteria first. In order to fully understand the correctness of a protocol analysis system, an understanding of what a protocol system is intended to do is needed. The scope criteria address what protocol systems can do and are therefore presented before the correctness criteria[3].

[3] The METHODS and RATINGS AND WEIGHTS sections will follow the same ordering as this section.

1. Scope

At the core of any system are the capabilities that the system supports and the systems underlying characteristics, we categorize a criterion of this nature a *scope* criterion. The scope criteria we present here are the core criteria for the presented evaluation methodology.

a. Type Support

The *types* that a protocol analysis system supports are inherently important in not only determining how to specify a protocol in the analysis system but also if the protocol can even be appropriately specified in the analysis system. Below is the list of types that are included in the evaluation methodology; this list was derived from [CLAR1997].

- Message
- Principal
- Nonce
- Timestamp
- Symmetric Cryptographic Key
- Asymmetric Public-Key Cryptographic Keys
- Symmetric Message Authentication Code (MAC)
- Asymmetric Signature/Verification Keys
- Text
- Number

Each type is considered separately from the others in this evaluation methodology. We classify the support that a protocol analysis system provides for a type in one of three ways: native, non-native, or unable. *Native* type support means that the analysis system implements support for that type by design, meaning that no additional enhancements need to be made by the user to make use of the type. *Non-native* type support means that the analysis system does not implement the type natively but there is a way to implement the type in the analysis system. This definition of non-native includes both ways that use other native types to simulate another type and third party work that

adds support for that type in the analysis system. A type that is not supported by the analysis system and is not implemented via extensions to the analysis system is labeled as *unable*. We admit that given these definitions it is possible that a type that has the possibility to be non-natively supported in an analysis system but has not yet been implemented would fall under the "unable" definition.

b. Operation Support

In addition to types, the *operations* that a protocol analysis system supports correlate to the capabilities of the analysis system. Just as with types, the operations supported by an analysis system are inherently important in not only determining how to specify a protocol in the analysis system but also whether the protocol can even be appropriately specified in the analysis system. Below is the list of operations that are included in the evaluation methodology; this list was derived from [CLAR1997].

- Symmetric Cryptography
- Symmetric MAC Generation/Verification
- Asymmetric Encryption/Decryption
- Asymmetric Signature/Verification
- Hashing
- Addition
- Concatenation

We consider the operations listed separately from each other, but group reciprocal operations, such as asymmetric encryption and asymmetric decryption, together. As with types, operations can be supported in one of three ways: native, non-native, unable. The definitions of *native*, *non-native*, and *unable* for operation support are the same as for type support.

c. Session Type

Session type refers to the breadth and depth of protocol instances that the protocol analysis system explores during the evaluation of the protocol. The session type criterion is used to evaluate the analysis system on the completeness of the protocol

instances it is able to explore during the evaluation of a communications protocol. Protocol analysis systems may be classified by one of three session types: fixed, bounded, infinite. The *fixed* set contains analysis systems that have a set limit to the number of states that will be explored during the evaluation of the communications protocol. The *bounded* set contains analysis systems that have the ability to explore states of the protocol up to the limitation of the test platform resources, such as main memory limitations. The *infinite* set contains analysis systems that have no limitations to the number of protocol states that are analyzed.

d. Theoretical Testable Protocol Characteristics

The main purpose of a protocol analysis system is to analyze a communications protocol in its ability to provide aspects of information security. An evaluation methodology designed for protocol analysis systems would not be complete without addressing the information security characteristics that a system analyzes. Below is a list of protocol characteristics that current protocol analysis systems address; the list is a union of characteristics we obtained through a literature review. We reviewed literature that discussed communication protocols [CLAR1996, NEED1978], literature about protocol analysis systems [GUTT2006, SAUL2001], and general literature about information assurance [CNSS4009].

- Confidentiality
- Integrity
- Authentication
- Availability
- Non-repudiation
- Round Efficiency

As with supported types we treat each of these characteristics separately within this evaluation methodology. The testability of each characteristic for a given protocol analysis system is categorized in one of three ways: native, non-native, or unable. Again, the definitions of *native*, *non-native*, and *unable* are consistent with the definitions presented for type support.

Of these characteristics, it is appropriate to explain the last two in greater detail as the first four are tenants of information assurance and hold a consistent meaning. *Non-repudiation* is defined by CNSSI 4009 as assuring that the sender of data is provided with proof of delivery and the recipient of data is provided with proof of the sender's identity. Non-repudiation cannot exist without data integrity and authentication, and we hold that it is a viable information security characteristic that can be addressed by a protocol analysis system.

We do not consider *round efficiency* a tenant of information assurance as it is specific to communication protocols. We consider round efficiency to fall under the availability tenant of information assurance. Some protocol analysis systems are able to make statements concerning the round efficiency of the communications protocol, SPEAR II is an example of one such system [SAUL1999, SAUL2001b, SAUL2003].

2. Correctness

With an understanding of what protocol systems can analyze, we can now discuss what criteria can be used to evaluate the *correctness* of a protocol analysis system. Protocol analysis systems, as with any function, have two important aspects that must be understood before the data involved can be transformed into information. When looking at a function in general, one is interested in the inputs and the outputs. The key input to a protocol analysis system is the specification of the communications protocol itself. The key output is the evaluation of the protocol returned by the analysis system. Table 1 depicts the possible combinations of results that a protocol analysis system can return when crossed with the security characteristic of the communications protocol. Table 1 is used for each of the testable protocol characteristics mentioned in the testable protocol characteristics criterion. The cells are labeled with an abbreviation for the protocol analysis system results followed by an abbreviation for the protocol. In general a protocol is either secure or insecure in regards to the testable protocol characteristic. A protocol is defined as secure (S) if it does not have a known flaw. A protocol is insecure (I) if a known flaw exists. The results returned from a protocol analysis system can be classified in one of four ways: secure, insecure with information, insecure without information, or inconclusive. We discuss the meaning of these classifications following the table.

		Testable Protocol Characteristic	
		Insecure	Secure
Protocol Analysis System Results	Insecure with Information	IW-I	IW-S
	Insecure without Information	IO-I	IO-S
	Secure	S-I	S-S
	Inconclusive	N-I	N-S

Table 1: Protocol Analysis System Results x Protocol Possibilities

An analysis system that returns an insecure result that is accompanied with amplifying information that shows how the protocol is insecure or presents an attack method against the protocol is classified as insecure with information (IW). If an analysis system returns an insecure result without any amplifying information or an attack method then the insecure without information (IO) classification is used. Secure (S) results are simply results from the protocol analysis system that state that the protocol is secure in regards to the associated characteristic. We consider inconclusive (N) results to be either a statement, or possibly a lack of statement, from the protocol analysis system that says that the tested characteristic of the protocol cannot be mathematically proved to be secure or insecure, or that the protocol analysis system fails to halt during the testing of the protocol. If an analysis system is unable to test a given characteristic then nothing can be said of which cell the system fits into; i.e., there is no cross product table for the analysis system in regards to the characteristic in question.

There are three cells that warrant further discussion: the IW-S, IO-S, and S-I cells. We will first discuss the S-I cell, as it is of interest within the evaluation methodology, the IW-S and IO-S cells have additional importance outside of the methodology. The S-I cell is evidence of a false positive. In the S-I case the protocol is known to be insecure but the analysis system returns results that say the protocol is in fact secure, thus the analysis system is incorrect.

The IW-S cell is the most important cell, this cell represents the situation where a protocol was previously thought to be secure but an analysis system returns results that

show how the protocol is actually insecure. When this situation arises there is immediate cause to notify others throughout the protocol analysis community and information security community in general. The IO-S cell is the cell of next importance; there are two cases that fall into this category; the first is if the analysis system is correct, the other when the analysis system is incorrect. In either case the human protocol analyzer will need to do further analysis to determine whether the analysis system is correct or not due to the lack of amplifying information provided by the analysis system. If the protocol can be shown to be insecure through further effort then this is also cause for notifying the community.

With an understanding of correctness classifications we can now discuss the correctness criteria. We present two correctness criteria: a *theoretical* criterion and an *observed* criterion. There is a need to include both criteria. The most general example of the need for both criteria is associated with the set of protocols that an analysis system will halt on. There might not be a way to define the set of protocols that a given analysis system will or will not halt on. For instance the theoretical correctness results might say that an analysis system will fall in one of the inconclusive cells (i.e., the analysis might not halt); whereas the observed correctness results would show that the analysis system does in fact halt on a specific protocol.

a. Theoretical Correctness

The *theoretical correctness* criterion is based on research results. This gives the theoretical correctness criterion the ability to provide results that might not be apparent from the observed correctness criterion testing; the observed correctness criterion will be discussed in the next subsection. The goal of the theoretical correctness research is to determine which of the cases the evaluated protocol analysis system fits into through research done by the evaluator. Each of the *testable protocol characteristics* discussed in the Scope section are considered in the theoretical correctness research. The details about the research to be performed will be discussed in the METHODS section.

b. Observed Correctness

The *observed correctness* criterion is based on experimental results returned by the protocol analysis system. As with the theoretical correctness criterion, the list of testable protocol characteristics previously presented will each be tested by the evaluator. In the METHODS section we will discuss how to perform the testing.

3. Performance

In any system there exist a set of limited resources that are of concern to the users of the system. We present a set of *performance* criteria for resources whose scarcity are often of concern in computer systems and by protocol analysis system users.

a. Execution Time

System users often ask the question, "How long is it going to take for Process A to complete?" The execution time criterion partly answers this question. The *execution time* criterion encompasses only the time the protocol analysis system spends on analyzing a given protocol. The method for testing execution time will sufficiently address and account for the issues of variability in execution time and will be discussed with the other testing methods in the METHODS section. The execution time criterion does not include any installation time, setup time, user interaction time, or user analysis time. The installation and setup times are essentially one time occurrences and are not factored into the evaluation methodology. The user interaction and analysis times are more appropriate to include in as a Usability criterion and are discussed with other Usability criteria.

b. Secondary Memory Requirement

One of the more critical requirements that any computer program has is its *secondary memory requirement*, or the amount of hard disk space that the analysis system requires. The continual growth in storage capacity will not change the need to address secondary memory requirement concerns. The obvious unit to measure secondary memory requirement*s* would be in bytes, as this is the standard unit of measure used in the computer industry as a whole.

c. Main Memory Requirement

Another resource that is of concern in computer systems is *main memory requirement*, or RAM requirement. As with the secondary memory requirement criterion, the appropriate unit of measure for main memory requirement is the industry standard byte.

4. Usability

We will now discuss criteria that we categorize as *usability* criteria. Some of the usability criteria we present are objective in nature while others are subjective. The usability criteria we present go beyond the simple classification of text or graphical based systems; a user can be more productive with an effectively designed text based interface than a poorly designed graphical user interface. We present criteria that are based on functionality, errors counts, and lengths of time. These criteria evaluate a system better than a simple interface classification.

We consider the first two usability criteria to be objective criteria and can be tested without the need for participants or test users. We consider the other six usability criteria subjective in nature and expect them to include participants in the testing methods. As before, we define the criteria here and present testing methods in the METHODS section.

a. Automation

Automation has been a part of computer systems since the introduction of batch programming. The problems solved by batch programming are not nearly as advanced as the problem of proving the security characteristics of a communications protocol, but none the less there are automated methods for proving such characteristics. The existence of automated protocol analysis systems validates the inclusion of the automation criterion. We consider three classes of automation: automated, automatable, and non-automatable. An *automated* system is a system that has an automated analysis mode built into the system by design. An *automatable* system is a system that does not have an automated analysis mode built in but the commands required to analyze a protocol could be automated *a priori* via a known automation tool. For example, if a

system required the same three steps to analyze any protocol but these steps are not automated within the analysis system and there exists a known tool to automate the steps then the protocol analysis system would be categorized as automatable[4]. Lastly, an analysis system that is not capable of being automated is categorized as *non-automatable*.

b. Specification Comments

Just as in computer code, comments are used inside a protocol specification to aid the reader in understanding. We consider three classes of comments: smart, flat, and unable. *Smart* comments are comments that have the ability to include links, such as hypertext, to information outside of the specification itself. *Flat* commenting systems are systems that do not afford the ability to include hot links within comments. A system that does not allow for any comments to be included in the specification classifies as *unable*.

c. Syntax Errors During Protocol Specification

The number of errors made while a protocol is being specified is indication of the usability of the system. We define syntax errors as syntactical errors that result from a user error while specifying the protocol. The exclusion of a semicolon at the end of a C++ statement is an example of a syntax error. The number of syntax errors is indicative of the analysis systems design to allow syntax errors to occur. The testing method for the syntax errors during protocol specification would need to address how this information is gathered in general for GUI and non-GUI analysis systems.

d. Structural Errors During Protocol Specification

The average programmer knows that just because a program compiles it does not mean that the program is correct; the same is true when specifying protocols. Just because an analysis system correctly parses a protocol specification does not mean that the protocol in question is correctly specified. A structural error is an error that results from the user specifying the protocol incorrectly. The number of structural errors is indicative of the analysis systems ability to support incorrectly specifying a protocol.

---

[4] One such automation tool is Expect [LIBE1995].

e. Specification Time

Above we presented a performance criterion that is used to measure how long it takes the analysis system to complete the analysis of the protocol, however, there is more to protocol analysis than just the tasks performed for the analysis system. In order for the analysis system to perform its tasks it must first be provided with a protocol specification. The specification time criterion is the time it takes a user to specify a protocol.

f. Results Analysis Time

The final step in using a protocol analysis system is to review and understand the results returned by the analysis system. This is sometimes a trivial task and sometimes a daunting task. The results analysis time criterion is the time it takes a user to understand the results returned by the system.

g. Participant Feedback

Feedback from the analysis system user has the ability to provide insight of the system that might not be apparent through the other methodology criteria. The analysis system user is able to explain their reasoning and thought process. The opinions of the system users are of value.

h. Experimenter Feedback

Participant feedback itself does not fully capture the information pertinent to system evaluation. Including experimenter feedback allows for a more complete analysis of the system under evaluation. For instance, a participant might report that they did not feel any frustration when using the system; whereas the experimenter observed frustrated behavior from the participant. There are a few of reasons as to why the participant would not report the frustrated behavior. The participant might not have realized they were frustrated or might have merely forgotten to provide that feedback. Additionally, it is possible that the participant might have a bias towards the system under evaluation or the evaluator himself, and would curve their feedback accordingly. An

experimenter is able to provide another view of the situation through experimenter feedback.

B. METHODS

In this section we present the methods to be used to measure the previously defined evaluation criteria. The testing methods fall in one of two classes: research based, or experiment based. The research based testing methods all follow the same steps, which we will discuss in the next paragraph. The experiment based testing methods are unique to the criteria being tested and will be discussed in detail with the sub-section for the criteria. We end this section with a discussion of the test ordering. Before we begin our discussion of the testing methods we present the research based testing method that is used to test a number of criteria.

Research based testing focuses on conducting a literature review; the goal of the literature review is to determine how the protocol analysis system fits the categories defined in the CRITERIA section. The first step in research based testing is to review the formal documentation provided by the developers for the analysis system. If a review of the formal documentation does not provide a clear determination as to how the analysis system should be categorized then the literature review will need to be expanded. The literature review should be expanded to include other scholarly articles and documentation not necessarily provided with the system. If an expanded literature review fails to produce a clear categorization, then as a last resort the protocol analysis system designers can be contacted[5]. If the system designers are unable to be contacted or are themselves unsure of how to categorize the analysis system, then the analysis system should be categorized with the lowest category[6]. For example, if the system could not be clearly categorized based on the literature review, the lowest category is *unable*, and the system designers cannot present a case as to why there system should be rated higher, then the system is categorized *unable*. Additionally, in a situation where information reviewed from one source conflicts with another the most current information should be used.

---

[5] We use the term *designer* to include the original and current maintainers of the system.
[6] As discussed in the RATINGS AND WEIGHTS section.

With our discussion of the common testing methods complete we can now begin to discuss each testing method in more detail; we begin with pre-testing.

1. Pre-Testing

Before testing of protocol analysis systems can begin a test platform must be setup and configured. We consider *pre-testing* to include activities that occur before the testing methods of the evaluation methodology are conducted. The first step in pre-testing is to determine which protocol analysis systems are to be evaluated and determine their minimal requirements. This will lead the evaluator to a list of minimal requirements that the test platform(s) must meet in order to successfully complete the evaluation. Decisions such as which host operating systems the analysis system is designed for, minimal main memory requirements and minimal secondary memory requirements are representative minimal requirements that must be met by the test platform(s). If it is determined that multiple test platforms are needed to conduct the evaluation every feasible effort should be taken to mirror the test platform configurations[7]. Once it has been determined what components will make up the test platforms they can be purchased and assembled upon receipt. After the test platform and components have been assembled the operating systems can be installed and configured. The next step is to install and configure the protocol analysis systems that are to be evaluated. Finally, the system and analysis systems configurations must be recorded to facilitate analysis needs. Any issues or deviations from standard configurations should also be documented to facilitate analysis needs. The system and protocol analysis system configurations for our test platform are presented in APPENDIX A:.

2. Scope Testing

a. Type Support

The testing method for the type support criterion follows the research based method. Each type in question is considered individually from the others. The goal

[7] This has recently become an easier task as the Macintosh operating systems now run on x86 architectures.

of the research is to determine the degree (native, non-native, or unable) in which the type is supported by the analysis system in question.

b. Operation Support

As with type support, the research based testing method is used for the operation support criterion. Each operation in question is considered individually from another. The goal of the research is to determine the degree in which (native, non-native, or unable) the type is supported by the analysis system in question.

c. Session Type

The testing method for the session type criterion also uses the research based method. The goal of the session type testing is to determine whether the protocol analysis system in question performs a fixed, bounded, or infinite state analysis.

d. Theoretical Testable Protocol Characteristics

The theoretical testable protocol characteristics criterion uses the research based testing method. As with the type support criterion, each of the testable protocol characteristics is considered separately. The goal of the research is determine the degree (native, non-native, or unable) in which that a characteristic is able to be tested by the analysis system.

3. Correctness Testing

a. Theoretical Correctness

As expected based on the name, the theoretical correctness criterion follows the research based testing method. The goal of the theoretical correctness testing research is to determine which of the eight cases from Table 1 the protocol analysis tool fits in for each of the testable protocol characteristics. If the scope research determined that the analysis system does not test for the characteristic in question then the lowest rating is used[8].

[8] As discussed in the RATINGS AND WEIGHTS section.

b. Observed Correctness

As with the theoretical correctness criterion, the goal of the observed correctness criterion is to determine which of the eight cases from Table 1 the protocol analysis tool fits in for each of the testable protocol characteristics. The observed correctness results are based on experiments rather than research. There are two additional points that must be discussed in regards to the observed correctness criterion: which protocols to implement, and how to comprise a single criterion rating for the analysis system. We present a set of three protocols that form the basis of determining the observed correctness criterion rating for a protocol analysis system. Each of the protocols should be implemented and run within the protocol analysis system; we discuss the rates for the correctness criteria in the RATINGS AND WEIGHTS section. In this evaluation methodology we implement the Needham-Schroeder [NEED1978], Needham-Schroeder-Lowe [LOWE1996], and Kerberos communication protocols [RFC4120]. The Needham-Schroeder protocol was chosen to test the analysis system for the possibility of a false positive[9]. The Needham-Schroeder-Lowe protocol was chosen to test whether the analysis system can prove that a two principal communication is secure, and to see if the analysis system can handle asymmetric cryptography. The Kerberos protocol was chosen to test the ability of the analysis system to handle a client-server protocol as well symmetric cryptography.

There are various ways in which results from the implemented protocols could be combined to determine a single observed correctness criterion rating. We use a minimum function to calculate the observed correctness result for a protocol analysis system under evaluation. The inputs to the minimum function are the cases from Table 1 that are determined for each of the implemented protocols based on the results returned from the analysis system. The output from the function is the case with the lowest rating[10]. The use of the minimum function assigns the lowest observed rating to the protocol analysis system, which is consistent with the need to make sound decisions.

---

[9] The Needham-Schroeder protocol is known to be flawed [LOWE1996] and is therefore a suitable choice to test for the possibility of a false-positive.

[10] As discussed in the RATINGS AND WEIGHTS section.

4. Performance Testing

In this section we present the testing methods for performance criteria; the methods we present are for our POSIX based test platform[11]. When other platform types are used in future evaluation these testing methods will need to be adapted and validated on other types of platforms. Such adaptations can be performed by the evaluators; we present a method for enacting such adaptations in the METHODOLOGY ADAPTATIONS section.

a. Execution Time

We mentioned when defining the execution time criterion that it was an imperfect unit of measure and that the variability of the measurements must be accounted for. This is done by running 51 trials; 51 trials were chosen based on the need to use a Gosset *t* model for data analysis[12]. The `time` tool is used to gather timing data[13]. If a trial took longer than 60 hours to complete the trial is terminated and the value of 60 hours is used. Execution time data should be gathered for each of the implemented protocols (NS, NSL, and Kerberos).

The systems tested already had automated modes of operation and did not require the use of automating tools such as Expect [LIBE1995]. If any of the systems to be tested and compared require the use of an automating tool then all the systems must be run under the same automating tool. Otherwise it would be inappropriate to compare data between systems that are not run under the automating tool and those that are.

b. Secondary Memory Requirement

The `ls` tool is used to gather secondary memory requirement data[14]. Data from the main directory and all subdirectories is gathered and totaled.

---

[11] APPENDIX A: lists our test platform specifications in detail.

[12] The use of the Gosset *t* model will be further discussed in the RATINGS AND WEIGHTS section.

[13] The scripts used to gather the timing data are included in APPENDIX B:.

[14] The scripts used to gather the timing data are included in APPENDIX B:.

c. Main Memory Requirement

The `time` tool is also used to gather main memory requirement data[15]. As with the execution time criterion, 51 trials were run as the Gosset *t* model is again used for data analysis. Additionally, data is collected for each of the implemented protocols.

5. Usability Testing

The Usability criteria consist of criteria that are purely objective and others that incorporate a due amount of subjectivity. As such there are two distinct types of testing methods presented, one for the objective criteria and another for the subjective criteria.

a. Automation

The automation criterion is an objective usability criterion that is suitable for testing via the research testing method. The goal of the automation research is to determine which of the automation categories the system under evaluation falls into.

b. Specification Comments

The question as to the kinds of comments allowed in the protocol specifications can also be answered via the research based testing method. As expected, the goal of specification comments research is to determine which type of comments the analysis system allows.

c. Subjective Usability Criteria

Instead of a detailed description of the testing methods for the subjective usability criteria we discuss some of the issues that should be addressed when the usability tests are designed and present our general ideas of how the testing could be conducted. Designing detailed subjective testing methods was beyond the scope of this project and should not be taken lightly.

The first issue that we discuss is determining the participant classes that are to be used in the testing methods. The needs of the evaluator will play a part in determining which class of participants should be used in testing methods. For instance,

[15] The scripts used to gather the timing data are included in APPENDIX B:.

if an educator is evaluating analysis systems to determine which system is best suited for classroom laboratory exercises, a “novice” participant class might be appropriate. On the other hand, a communications system manager would not be expected to assign the task of analyzing protocols to “novices,” and would instead use a participant class with a level of experience closer to the analyzers of his communications system.

The human nature of the participants appropriately limits the usability testing methods to be non-destructive in nature [NIEL1993]. Destructive test methods are commonly applied to hardware products, for example drop testing a laptop. The use of test participants prohibits the use of such destructive testing methods. Although protocol analysis has caused headaches in its own right, we do not expect there to be an issue in designing non-destructive testing methods. Nielsen points out guidelines that should be followed when conducting user testing.

As with participants there are various levels of expertise in experimenters. The testing methods need to account for the experience of the experimenters. The experimenters should be familiar with not only the testing methods but also the analysis system(s) being tested. Nielsen points out that extensive system knowledge is necessary because the experimenter needs to understand what participants are doing while they are using the system. This knowledge allows the experimenter to make reasonable inferences about the users.

Perhaps the most significant non-human elements of the usability testing are the test tasks. The tasks must be designed to meet a number of criteria. Again, the test tasks must be non-destructive. The instructions for the tasks should be clear enough for the participants to understand them. A number of participants not understanding a task is indicative of an unclear task. Unclear tasks will require the participants to seek the help of the experimenter. This has two negative effects; 1. The participant will begin to feel frustrated, and 2. The results will begin to be for the experimenter and not the participant. The tasks need to be appropriately sized as well. The tasks should be small enough to be completed in the allotted time, but not too small as to be trivial [NIEL1993]. Nielsen stresses that the first task should be simple enough for any novice user to complete, thus raising the confidence of and relieving tension in the participant.

Nielsen points out and discusses four typical stages of usability testing: preparation, introduction, testing, debriefing. The preparation stage is similar in concept to the pre-testing we previously discussed. In the preparation stage the test room and equipment is prepared. The evaluator and experimenters also review the testing methods, system(s) to be evaluated, and discuss any concerns they might have. After the testing environment has been prepared the experimenters can begin the introduction. In the introduction the experimenters will cover the concepts of the tests with the participants. After all concerns of the participants have been addressed, the actual testing can begin. Once the testing has completed the participants and experimenters should be debriefed and given a chance to provide feedback.

The last significant topic that is appropriate to discuss is the feedback mechanism for both the experimenters and participants. There are two main ways in which feedback can be provided: questionnaires, and interviews. Questionnaires and interviews indirectly test the analysis system, meaning that questionnaires and interviews focus on the opinions of the participants [NIEL1993]. This is contrary to the direct testing methods presented for the scope, correctness, and performance criteria that focus on the analysis system itself. Both questionnaires and interviews allow the evaluators to get feedback from the participants. A key issue in using questionnaires or interviews in an evaluation methodology is that the questions must be determined and standardized before hand. This does not mean that open ended questions cannot not be included in either a questionnaire or interview session. This does, however, mean that exploratory interviews should not be used; i.e., the interviewer should have a list of questions that should be discussed in the interview session, as opposed to beginning the interview without knowing what the interviewer is looking for.

6. Post-Testing

After all of the testing methods have been completed the next step is to determine the overall score for the system under evaluation. This is accomplished by inputting the results from the run testing methods into the overall weighting function. The overall weighting function will be further discussed in the RATINGS AND WEIGHTS section but it is sufficient at this point to mention that the function returns the overall evaluation

result, a numerical score, for the protocol analysis system under evaluation. Afterwards the returned evaluation can be analyzed. As with any analysis the more data that is collected increases the value of the analysis performed. Analyzing the results for a single system will provide valuable analysis for that specific system. When the evaluation results from multiple systems are compared, the analysis performed can provide even greater value to the evaluator. The final step in the evaluation process is a feedback loop, where the evaluation analysis serves as a means to adapt the analysis systems to better address the needs of the evaluator. A newly adapted analysis system can then be evaluated again to see the results of the system designers' efforts.

7. Test Ordering

We now present order in which to run the evaluation testing methods. Figure 4 depicts the required test ordering. There are only two requirements that the test ordering must meet. The first is that the pre-testing must occur before the criteria testing methods and before the post-testing. The second requirement is that the post-testing must be performed after the pre-testing and after the criteria testing methods. The specific ordering of the criteria test methods is left up to the evaluator. The lack of strict ordering requirements allows the evaluator to order the tests based on the situational needs. Figure 4 also shows that the criteria tests can be run in parallel.

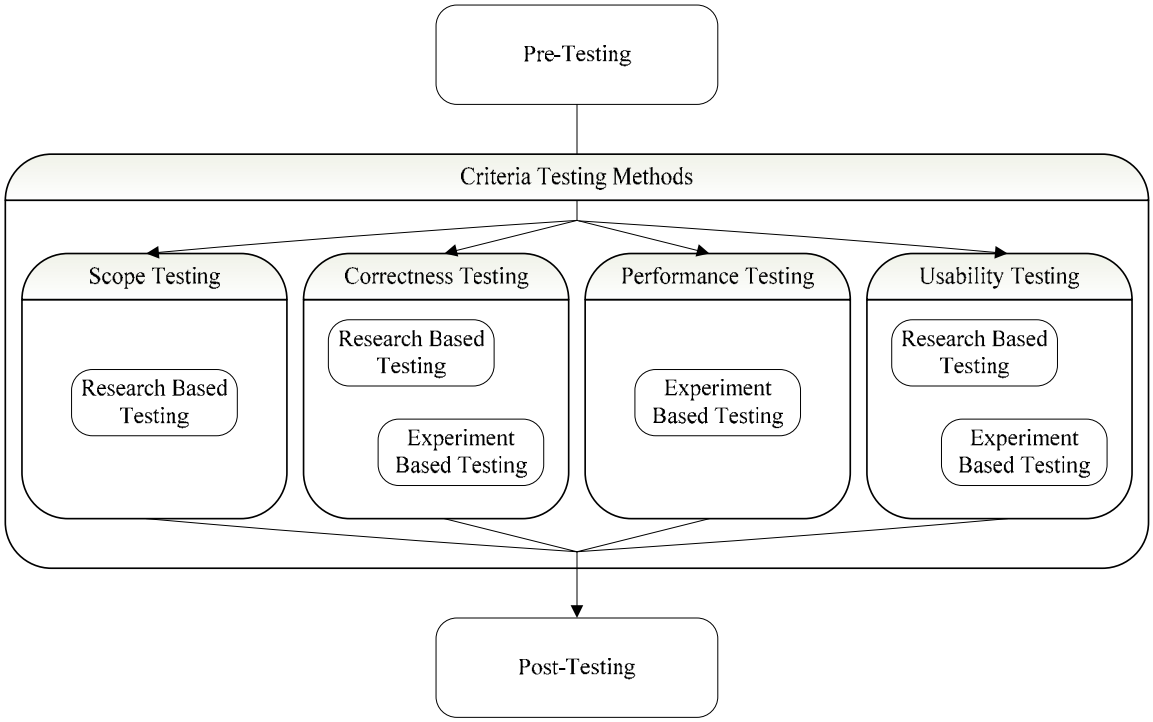

Figure 4: Required Testing Order

Figure 5 presents a recommended testing order. The order presented in Figure 5 meets the two test ordering requirements. We recommend that the research based testing methods be performed the experiment based testing methods. Performing the research based testing methods before the experiment based testing methods gives the evaluator a better understanding of what to expect when running the experiment based tests. For example, through research an evaluator could determine that the class of protocols that the analysis system halts on is unknown. This information could be used by the evaluator to help determine whether the system is taking a long time to complete the analysis or that the system will likely not halt. The ordering within in the research and experiment based boxes in Figure 5 are also recommendations; i.e., it is recommended that the experiment based correctness and performance testing be completed for usability testing. As before, this ordering allows for insight during the usability testing derived from the correctness and performance testing.

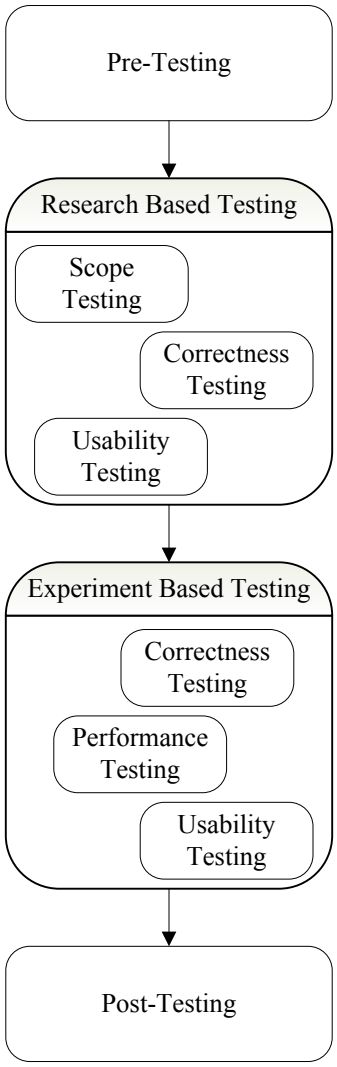

Figure 5: Recommended Testing Order

C. RATINGS AND WEIGHTS

The weights and ratings that make up any evaluation methodology are arguably the most interesting and contested. The ratings are the schemes used within each of the criteria. The weights are used in the overall weighting function to stress the importance of certain criteria over other criteria. The CRITERIA and METHODS sections were objective in nature, whereas this section is subjective in nature. The ratings and weights we present here are an example tailored to the needs a communications system manager.

The ratings and weights can be tailored to better meet the needs of a different evaluator[16]. For instance, the ratings and weights can be tailored to meet the needs of an educator interesting in using protocol analysis systems in a classroom laboratory setting. The ability to tailor the evaluation methodology to the needs of the evaluator is one of the key aspects of this evaluation methodology.

In this section we first present the rating system that we used within each criterion. We then present the weight that we assigned to each criterion to determine our overall evaluation result of the protocol analysis system.

Table 2 depicts the rating order and the point values associated with a rate for criteria that use a rating system as a scoring mechanism[17].

Rating	Ordering	Point Value
High	Highest	9
Medium		3
Low		1
Unacceptable	Lowest	-1

Table 2: Criterion Rating

The immediate question that arises is how the point values listed were chosen. The ordering of the point values is not likely to be questioned as the highest order rating also has the highest associated point value and the lowest ordered rating has the lowest associated point value. It is interesting to note that the unacceptable rate is assigned a negative point value; this not only penalizes an analysis system for an *unacceptable* rating but opens the possibility for an analysis system to achieve an overall negative score. The point values are derived from the Quality Function Deployment (QFD) concept [HJOR1992], where the point values 9, 3, and 1 are used. QFD is a design process whose goal is to incorporate the needs of the customer throughout the product design process. The criteria of an evaluation methodology can be viewed as the "what's" of QFD planning matrices, such as House of Quality (HoQ) diagrams. The ratings used

[16] Tailoring the ratings and weights used in the evaluation methodology is considered an adaptation to the methodology, and is discussed more in the METHODOLOGY ADAPTATIONS section.

[17] These point values were chosen before any testing methods were run or results analyzed.

can be thought of as the corresponding "how's." Incorporating concepts such as QFD into the design of an evaluation methodology extend the usefulness of the methodology. We intend this evaluation methodology to not only be used by users of protocol analysis systems but also by the protocol analysis system designers. Showing how this evaluation methodology can be mapped to QFD concepts can aid developers in understanding how to better design protocol analysis systems to meet the needs of the analyzer.

1. Scope

a. Type Support

Table 3 presents our example rates associated with the classifications for types supported by analysis systems. A type that is natively supported by a protocol analysis system is rated high because of the minimal effort required to implement concepts that require the use of the type in question. A non-natively supported type is rated medium due to the increased effort that is required to implement concepts that require the type in question. If a type is unable to be supported in a protocol analysis system a low rating is assigned. As previously mentioned, each type is considered individually; the weightings used between criteria will be discussed later in Section III.C.5 Overall Weightings.

Type Support	Rating
Native	High
Non-Native	Medium
Unable	Low

Table 3: System Type Support Criterion Rating

b. Operation Support

The example operation support rating system mirrors the type support rating system as depicted in Table 4. The reasoning behind the operation support ratings is the same as those presented for the type support rating system.

Operation Support	Rating
Native	High
Non-Native	Medium
Unable	Low

Table 4: System Operation Support Criterion Rating

c. Session Type

Table 5 presents the example rating system for the session type criterion. A system that supports infinite sessions has the possibility to explore more states than either a bounded or fixed analysis system. Similarly, a bounded system has the possibility to explore more states than a fixed protocol analysis system. The ratings in Table 5 reflect the above statements.

Session Type	Rating
Infinite	High
Bounded	Medium
Fixed	Low

Table 5: System Session Type Criterion Rating

d. Theoretical Testable Protocol Characteristics

The rating system employed for the testable protocol characteristic criterion is similar to the other criteria that use the native, non-native, and unable classes. The same reasoning presented in the type support rating system discussion applies to the example theoretical testable protocol characteristics rating system depicted in Table 6.

Testable Protocol Characteristic	Rating
Native	High
Non-Native	Medium
Unable	Low

Table 6: System Theoretical Testable Protocol Characteristics Criterion Rating

2. Correctness

The correctness of an algorithm is often its most important characteristic. It is often the case that an algorithm will not be used if it is found to be flawed. However, our example evaluation rating scheme does allow for the acceptance of a system with the presence of flaws. Table 7 summarizes the approach taken by our example communications system manager evaluator; this approach is the basis for both the theoretical and observed correctness criteria. As a reminder, the testing methods presented in the METHODS section assigned the lowest rating based on the results of the tests. The use of the lowest rate is consistent with the needs of a communications system manager.

		Protocol	
		Insecure	Secure
System Results	Insecure with Information	High (IW-I)	High (IW-S)
	Insecure without Information	Medium (IO-I)	Medium (IO-S)
	Secure	Unacceptable (S-I)	High (S-S)
	Inconclusive	Unacceptable (N-I)	Low (N-S)

Table 7: System Correctness Criterion Rating

Any scenario that has the potential to lead a manager to an unsafe decision is deemed unacceptable and the protocol analysis system will be penalized in this evaluation. It is clear that regardless of who is using the protocol analysis system or for what purposes, a system that returns a false positive is unacceptable, as depicted in Table 7. A false positive from an analysis system could lead the communications system manager to make an unsafe decision, which is a direct contradiction of the responsibility of a manager. A case that is also grounds for giving the analysis system an “unacceptable” correctness rating is if the system, through the evaluation, is determined to fit the N-I case from Table 1. This situation also has the possibility of leading the manager to make

an unsafe decision and is thus rated accordingly. The N-I case is similar to the S-I case, in that the analysis system fails to identify a known flaw in the protocol. The lack of identifying a flaw in the N-I case will not lead the manager away from making an unsafe decision.

A protocol analysis system is assigned a low rating if it is determined by the testing method to fit in the N-S case. Analysis systems that fit this scenario will not lead to an unsafe decision but will not aid in the decision making process either. This point is an area in which this rating system may differ from a rating system designed to meet the needs of a different evaluator. By focusing on the managers view this rating system favors systems that aid in the decision making process, over absolute correctness. The "medium" correctness criteria rate is evidence of this fact.

An analysis system is assigned a medium correctness rating if the testing method determines that the analysis system fits into IO-I or IO-S case. As previously mentioned the IO-S case could be a false negative. The "medium" rating of a system that has been determined to allow for false negatives is another point in which this rating system may differ from schemes designed for other evaluators. From the decision making perspective a false negative will not lead to an unsafe decision and therefore a system that fits this category will not be penalized by this methodology. On the other hand, the fact that the analysis system does not provide any amplifying information limits the use of a higher rating.

The final correctness criteria rating a protocol analysis system can achieve is "high." This is achieved when the methodology determines that the analysis system fits in any of the IW-I, IW-S, or S-S cases. In the IW-I case the protocol analysis system correctly identifies the protocol as insecure and provides amplifying information as to why the protocol is insecure, thus greatly aiding in the decision making process. As mentioned in the CRITERIA section, the IW-S case provides information that is not only pertinent to the evaluator (communications system manager in our case) but to the protocol analysis community in general. In the S-S case the analysis system correctly identifies the protocol is secure, again aiding in the decision making process.

3. Performance

a. Execution Time

Before we can discuss the rating system used for the execution time criterion we must first address the variability of the data collected by the testing method. In Figure 6 we present the formula used to determine the high end (CI^+), or longest running time, of the 95% confidence interval. Our rating system first uses statistics (Gosset's *t* model) to place an upper-bound on the mean execution time, then translates this upper-bound into a rating. Gosset's *t* model was chosen over other models because the actual population mean is not known; instead only a sample mean is available. Gosset's *t* model appropriately accounts for the variability that occurs between separate trials. The high value (CI^+) was chosen as there is only a 2.5% chance that the actual population mean will be above the calculated high value. The value for t_α was retrieved from a standard statistics table in [DEVE2005], and correlates to a 95% confidence interval based on the number of trials (n) ran in the test method. The sample mean is denoted by $\bar{X}$ and is the average of the trial data, each trial is represented as X_i. The SE function is known as the standard error function. The *s* formula is the standard formula for calculating the standard deviation of a sample.

$$CI^+ = \bar{X} + t_\alpha \, \mathrm{SE}\left(\bar{X}\right)$$

$$\bar{X} = \frac{\sum_{i=1}^{n} X_i}{n}$$

$$n = 51$$

$$t_\alpha = 2.009$$

$$\mathrm{SE}\left(\bar{X}\right) = \frac{s}{\sqrt{n}}$$

$$s = \sqrt{\frac{\sum_{i=1}^{n} (X_i - \bar{X})^2}{n-1}}$$

Figure 6: System Execution Time CI^+ Calculation

With an understanding of how variability in execution time was accounted for we can now discuss the rating system for the execution time criterion. The rating

system used is similar to rating systems used for other criteria; the difference is that there are more ratings used, as evident in Table 8. The CI^+ value serves as the input into the execution time rating system. In general, the time intervals are based on common office time intervals; these time intervals are consistent with the needs of our example evaluator, the communications system manager. The high rating is along the lines of a trip to the coffee machine or a stretch break. The medium-high rating is equivalent to a lunch break or meeting. The medium value is equivalent to a half-day. The medium-low rating is representative of a process that is run overnight. The low rating is the time frame of a process that is allowed to run over the weekend. If an analysis system fails to return results or takes longer than 60 hours to return results a *nil* rating is assigned.

CI^+	Execution Time Rating	Point Value
CI^+ < 5 min	High	9
5 <= CI^+ < 60 min	Medium-High	5
1 hr <= CI^+ < 4 hr	Medium	3
4 hr <= CI^+ < 12 hr	Medium-Low	2
12 hr <= CI^+ < 60 hr	Low	1
CI^+ >= 60 hr	Nil	0

Table 8: System Execution Time Criterion Rating

b. Secondary Memory Requirement

In Table 9 we present the protocol analysis system Secondary Memory Requirement (SMR) criterion ratings. The data collected from the testing method is summed to form a grand total SMR value. Regardless of storage capacity, systems that require less main memory are preferred to those that require greater main memory requirements. As main memory storage capacity increases it is likely that the boundary values used as separation points between the ratings will increase, we discuss these and other criteria adaptations in the METHODOLOGY ADAPTATIONS section. The main memory requirement rating divisions are based on current day disk storage sizes.

Secondary Memory Requirement (SMR)	Rating
MMR < 512 MB	High
512 MB <= MMR < 1024 MB	Medium
MMR >= 1024 MB	Low

Table 9: System Secondary Memory Requirement Criterion Rating

c. Main Memory Requirement

The same equations and values used for the execution time criterion used for the execution time criterion are used for the main memory requirement. This is because the Gosset *t* model was again needed to account for the variability between the trials, and the same number of trials was run for the two criteria. The obvious exception is that the maximum memory usage data served as the X_i values, as opposed to the execution time data.

Table 10 presents the boundary values and associated rates for a protocol analysis system's main memory requirements. As with the secondary memory requirement criteria systems that have lower main memory requirements are rated higher than those with greater main memory requirements. As with the secondary memory requirement the boundary values presented in Table 10 are expected to be adjusted as the storage capacity of main memory shifts over time. The rating divisions for the main memory requirement are based on current day common DIMM sizes. If the analysis system must be stopped before the protocol analysis completes then the lowest rating should be used, and "Failed to Halt" should be used in data reports.

Main Memory Requirement (MMR)	Rating
SMR < 512 MB	High
512 MB <= SMR < 1024 MB	Medium
SMR >= 1024 MB	Low

Table 10: System Main Memory Requirement Criterion Rating

4. Usability

a. Automation

Table 11 lists the rating system for the automation criteria. An automated analysis system is desired by many users in the protocol analysis field, to include a communications system administrator. An analysis system that is automated is given a high rating. A protocol analysis system that is automatable is given a medium rating. Lastly, a low rating is assigned to analysis systems that are non-automatable.

Automation	Rating
Automated	High
Automatable	Medium
Non-Automatable	Low

Table 11: System Automation Criterion Rating

b. Specification Comments

In Table 12 we present the specification comments criterion rating scheme. Smart comments provide greater functionality than flat comments and are rated higher than flat comments. Similarly, flat comments provide commenting functions that are not present in systems without commenting features. Therefore, smart comments, flat comments, and no comments are assigned the ratings of high, medium, and low respectively.

Specification Comment Type	Rating
Smart Comments	High
Flat Comments	Medium
No Comments	Low

Table 12: System Specification Comments Criterion Rating

c. Subjective Usability Criteria

At this point in our research we do not have enough material to discuss rating systems for the subjective usability criteria. Specifically, more can be said about the subjective criteria rating systems once the testing methods are presented in a more formal manner.

5. Overall Weightings

Now that we have discussed the individual rating system used by each of the criteria the manner in which the overall score is determined can be discussed. The weightings used between the criteria serve as a way to signify which criteria are more important to the evaluator than others. It is important to note that extensive gathering and compiling of data before the overall weightings are defined leaves the weightings open to excessive bias. It is very hard to remove all bias from an evaluation especially in regards to the weightings used between criteria. Defining the weightings after significant data are gathered or any data are analyzed results in a higher probability of injecting bias into the evaluation than is necessary. If such an action were to take place the weightings would need to be thoroughly reviewed and determined to be unbiased. We realize that some degree of testing and prototyping is necessary in any project, and that such actions risk introducing bias into the project. We took great measures during the prototyping of the methodology in order to limit bias. For example, we did not perform any data analysis until after the overall weightings had been determined. Additionally, during the design of the document ultimately used to track evaluation data, dummy values were used instead of actual test data. Further, the experimental testing methods were validated on an earlier version of CPSA than was used in the evaluation[18].

We present the discussion of the overall weightings by the criteria categories; the weightings discussed are used in a collective manner to weight all the evaluation criteria amongst them selves. Our example weightings are based on the needs of a communications system manager. The discussions are presented in the same order in which the criteria were previously presented. Each criterion is independently assigned a weighting factor; the score for each criterion is calculated by multiplying the weighting factor by the associated rating factor discussed previously. The scores for the criteria are summed together to make the overall evaluation result of the evaluation methodology. The low and high scores are easily calculated and serve as basis points to be used by the evaluator. Inputting the high rating point values for all criteria into the overall weighting

[18] CPSA v0.70 was used in the validation of the experimental testing methods, whereas CPSA v0.81 was used in the actual evaluation process.

function results in a maximum achievable score, our values result in a maximum score of 1962. Inputting the low rating point values into the overall weighting function results in a minimum score, our values result in a minimum score of 81.

a. Scope Criteria Weightings

Table 13 presents the weighting assignments for the scope criteria. The scope criteria address the fundamental capabilities of a protocol analysis system. Thus, in general the scope criteria are assigned high weighting factors by our example evaluator.

The value of a protocol analysis system is highly dependent on the class of protocols that can be specified in the system. The types and operations that are available in the analysis system are directly correlated to the ability to specify a protocol in the system. In practice there are types and operations that can be used to simulate other types and operations. The types and operations that are weighted with a 9 are crucial to being able to specify even the most basic protocols. The types and operations with a 3 are typically implemented through the user of other types and operations, such as those weighted with a 9. The types and operations weighted with a 1 can also be implemented through the use of other types and operations. Again, these weightings are based on the needs of a communications system manager.

The type of session that is supported by the analysis system plays a large role in the depth of analysis that the system performs. Therefore, the session type is given a weighting factor of 9.

As mentioned at the beginning of this work, we are interested in protocol analysis systems designed to analyze the security characteristics of communication protocols. Confidentiality, integrity, and authentication are the three security characteristics that security protocols address; and are therefore weighted with a 9. The concept of non-repudiation is addressed by some protocols in some fashion, but not to an extent that the full definition of non-repudiation is achieved. The current lack of protocols that attempt to meet the full definition of non-repudiation limited us to weight non-repudiation at 3, as opposed to a higher value. Security protocols rely on low layer protocols to address availability; for example, the Kerberos protocol is implemented on TCP/IP based networks throughout the world. Since security protocols are not designed

to address availability concerns the availability criteria is weighted low, a 1. The round efficiency criterion is also weighted with a 1, this is due to the general lack of concern in regards to how round efficient protocols are. Saul points out that protocol efficiency has not been an issue of great concern due to the small number of steps in communication protocols and a communications system manager is not overly concerned with the efficiency of the security protocols on the network [SAUL2001b].

Criteria		Weighting
Type Support		-
	Message	9
	Principal	9
	Nonce	9
	Timestamp	3
	Symmetric Cryptographic Key	9
	Asymmetric Cryptographic Keys	9
	Symmetric MAC	3
	Text	3
	Number	1
Operation Support		-
	Symmetric Cryptography	9
	Symmetric MAC Generation/Verification	3
	Asymmetric Encryption/Decryption	9
	Asymmetric Signature/Verification	3
	Hashing	9
	Addition	3
	Concatenation	3
Session Type		9
Theoretical Testable Protocol Characteristics		-
	Confidentiality	9
	Integrity	9
	Authentication	9
	Availability	1
	Non-repudiation	3
	Round Efficiency	1

Table 13: Scope Criteria Overall Weightings

b. Correctness Criteria Weightings

The overall weightings for the correctness criteria are presented in Table 14. The correctness weightings are the same as the theoretical testable protocol characteristics, and follow the same reasoning.

Criteria		Weighting
Theoretical Correctness		-
	Confidentiality	9
	Integrity	9
	Authentication	9
	Availability	1
	Non-repudiation	3
	Round Efficiency	1
Observed Correctness		-
	Confidentiality	9
	Integrity	9
	Authentication	9
	Availability	1
	Non-repudiation	3
	Round Efficiency	1

Table 14: Correctness Criteria Overall Weightings

c. Performance Criteria Weightings

Table 15 depicts the overall weightings for the performance criteria. The performance criteria of greatest interest to our example evaluator are the execution time criteria. However, when compared to the other criteria of the evaluation methodology the execution time criteria are only weighted with the moderate factor of 3, because analysis system execution time is not the primary concern of a communications system manager. The memory requirement criteria are of even lower concern to the communications system manager. As the capacity of memory increases and the price per GB decreases the

memory requirements of software becomes less important. The current capacities and prices are already at levels that warrant the low weighting factor we use.

Criteria		Weighting
Execution Time		-
	NS Protocol	3
	NSL Protocol	3
	Kerberos Protocol	3
Secondary Memory Requirement		1
Main Memory Requirement		-
	NS Protocol	1
	NSL Protocol	1
	Kerberos Protocol	1

Table 15: Performance Criteria Overall Weightings

d. Usability Criteria Weightings

The objective usability criteria weightings are presented in Table 16. Both are given a moderate weighting value of 3. We agree that the usability of any system is important but that in general the scope criteria are of greater importance than system usability.

Criteria	Weighting
Automation	3
Specification Comments	3

Table 16: Objective Usability Criteria Overall Weightings

D. METHODOLOGY ADAPTATIONS

As previously mentioned, the components of the evaluation methodology were presented in a modular manner in part to facilitate adaptations to the methodology. In this section we will discuss some of the adaptations that we have envisioned. We agree that there are valid reasons for adapting the methodology beyond the ones we discuss here. Any adaptation to the evaluation methodology should be appropriately documented.

1. Criteria

In the criteria section we presented a set of criteria and discussed why they are appropriate to evaluate protocol analysis systems. We understand that the set presented might not address all of the interests of every evaluator. Additionally, we expect that future protocol analysis systems will require more or different criteria to best evaluate and compare analysis systems.

The first step in adding a criterion to the evaluation methodology is to review the current criteria in the evaluation methodology. This review serves two purposes. The first is to ensure that the proposed new criterion is not already included in the methodology as a single criterion. This relates back to the concept that each criterion be independent of others. Secondly, the review serves to determine if the data already gathered for multiple criteria in the evaluation methodology can be combined to answer the question posed by the proposed criterion.

Once it has been determined that there is a need for the proposed criterion it can be determined in which category that the new criterion belongs in. It is possible that the new criterion will require the addition of a new criteria category; we will discuss the adding of a criteria category later. After the new criterion has been placed in the appropriate category the testing methods can be determined for the desired test platforms. The order in relation to other testing methods that the new testing method should be run in must also be determined. There might not be a specific point in the overall evaluation that the new testing method should be run.

There is one more area that must be addressed before the new criterion can be added to the evaluation methodology. Before the new criterion can be added the rating mechanism for the criterion must be defined. Additionally, the criterion must be weighted in the overall weighting function.

After the weightings have been defined the new criterion can now be used in the evaluation methodology. If the new criterion is truly independent of the other criteria then the entire methodology does not need to be rerun on already evaluated protocol analysis systems. Instead, only the method pertinent to the new criterion needs to be run. The data

collected from previous evaluations can be used to determine the new overall evaluation that includes the weighting method with the new criterion.

Adding a new criteria category is not nearly as involved as the addition of a new criterion, as the adding of a new category is more of an administrative task. However, there is still a procedure to follow. Just as with adding a new criterion, the first step in adding a new criteria category is to review the evaluation methodology. The reason for this review is to determine whether the topic that the new category is to encompass is not already covered by another category. Once it has been determined that there is a need for the proposed category the category is added to the evaluation methodology. The adding of a new category to the evaluation methodology is mainly an organization issue, so the documentation that discusses how to perform the methodology should be updated to include the new methodology.

2. Methods

There are three adaptations that we envision regarding the testing methods. The first two involve the testing methods themselves; the third involves the order in which the methods are performed. We will discuss all three in this section.

The first adaptation we envision is the adapting of the testing methods to other platforms. The testing methods that are likely candidates for this type of modification are the performance criteria testing methods. In this project we presented POSIX based testing methods for the criteria. Specifically, our testing methods were run on a Fedora Core 5™ based platform[19]. Minor adaptations to the methods we presented in the METHODS section might need to be made if another POSIX operating system is used. It is likely that more than minor adaptations will be needed if the testing methods are ported to a completely different platform, such as an Alpha based platform. Minor adaptations should not require modifications to other parts of the evaluation methodology. The first step in adapting a testing method is to review the testing method being ported to understand what the method is providing and how the method works. This review should help in the design of the adapted testing method. The next step is to review the

[19] The specific details about the test platform configuration are presented in APPENDIX A:.

functionality that is present on the new test platform, in order to see how the testing method could be implemented on the new platform. After the initial reviewing is done the draft version of the testing method can be constructed. The draft version of the testing method should then be reviewed, tested for correctness, and modified as needed. Once the adapted method is stable it can then be incorporated into the evaluation methodology.

The second reason we envision for adapting a testing method is more general than adapting the method for a different platform. The testing method itself can be modified for a number of valid reasons. A likely reason that a testing method would be altered is if a better method to gather the data required by the criterion is found. An example would be to use a resource analysis tool that provides better data than the currently used tool. Changing the manner in which the usability criteria are evaluated, such as changing from the use of a questionnaire to using observing proctors, is a specific example of a modification that fits the second reason. It is possible that a modification of this nature will require changes to other parts of the evaluation methodology, such as a change in the order the testing methods are performed. A testing method modification might also require a modification to the rating mechanism for the criterion, which we will discuss in the next section. As with the other modifications discussed the first step is to review the current testing method to see what portions of the method need to be modified. It is possible that some portions of the testing method might not need to be changed. If appropriate, the functionality of the test platform should also be reviewed to determine how to best implement the new testing method. The next step is to construct a draft version of the new testing method. Before the new test method is incorporated into the evaluation methodology it should be reviewed, tested for correctness, and modified as needed. The order of the testing methods should also be reviewed to determine if there is a more appropriate testing order given the incorporation of the new testing method.

Lastly, an update to an existing test method is not the only reason that the testing order might be modified. Logistical reasons or other limiting factors might justify a modification to the order in which the testing methods are run. We presented a nominal testing order that is flexible enough to allow for a variety of specific testing orders. If a one-time modification to the testing order is made in an evaluation session then it is

sufficient to note the modification in the report of the evaluation results. In the case of a permanent modification to the testing order more effort is required. As with other modifications discussed the first step is to review the current testing order described by the evaluation methodology to determine if the proposed permanent adaptation is needed or a one time adjustment is appropriate. If it is determined that the new order is needed a draft version of the new ordering should be produced next. The draft ordering should then be reviewed, tested for correctness, and modified as needed. Finally, the new test ordering can be implemented into the evaluation methodology.

3. Ratings and Weights

The most interesting and powerful adaptations to the evaluation methodology will likely occur within the weights and ratings. The ability to adapt this methodology to best suit the needs of the evaluator is one of the key strengths of this evaluation methodology. We will first discuss why and how the criteria ratings can be modified and then discuss why and how the overall weightings can be modified.

We previously mentioned that the rating mechanism of a criterion should be reviewed when the testing method is modified. We envision another reason for modifying ratings and a way to compare previously collected evaluation data with new evaluation data. The most likely reason for criteria rating modifications, specifically performance criteria, is the passing of time. We mentioned that there will always be a limiting of resources that a computer system must conform to. However, these limits continue to be increased so the rating mechanisms of today might not be suitable in the future. For example, we presented a boundary of 1 GB in the main memory requirement criterion, in five years it is likely that a 1 GB boundary will be too low. Instead of using the older rating limits the limits should and can be updated. As expected the first step in modifying a rating mechanism is to review the current rating mechanism. The purpose of this is to determine what modifications are needed before the draft version of the new rating mechanism is produced. The draft version can then be reviewed, tested for correctness, and adjusted as needed. Finally, the new rating mechanism can be incorporated into the evaluation methodology. In the main memory example we presented above it would likely be the case that data used in an older version of the rating mechanism could be

used in the new rating mechanism without the need to run the test method again. The old data would simply serve as an input to the new rating mechanism and the overall evaluation of the protocol analysis system updated as well. This provides a clear way to compare earlier data to newer data, where the most current and same rating mechanisms are used on all data and the testing methods are not unnecessarily rerun for older data.

The ability to modify the overall weightings of the evaluation methodology allows the methodology to be adapted to the needs of a wide variety of evaluators. The weightings we presented are consistent with the needs of a communication system administrator. The specific needs of a system administrator are not inline with the needs of other protocol analysis system users. For instance, the weightings can be altered to better address the academic needs of an educator in the classroom. The first step is to review the weightings in the evaluation methodology to see which weightings should be adjusted. Then the draft version of the new overall weightings can be produced. The draft version should be reviewed to see that the evaluator needs are met and adjusted as appropriate. Finally, the overall weightings can be implemented into the evaluation methodology. It should be noted that we omitted the testing of the new weightings from this modification procedure. We did this to avoid any biasing that might occur if actual data was used to justify the appropriateness of the weightings. Using data from actual protocol analysis systems would lend the evaluator to tune the weightings to show a preference to a system that the evaluator already prefers.

IV. EXAMPLE EVALUATIONS

In this section we present the results from running the evaluation methodology presented in Chapter III. Each discussion begins with a background of the analysis system and then discusses the evaluation results. The results data will be presented and discussed in the same order that the evaluation methodology criteria were presented in Chapter III for consistency.

A. CRYPTOGRAPHIC PROTOCOL SHAPES ANALYZER

The Cryptographic Protocol Shapes Analyzer (CPSA) is a protocol analysis system designed by The MITRE Corporation for the National Security Agency. CPSA is based on strand space theory [THAY1999]. CPSA is a text-based system designed for use on POSIX based platforms and can be used in either an interactive mode or batch mode. Our experiments used CPSA in the batch mode.

1. Scope

a. Type Support

Table 17 shows the results from the research conducted during the review of CPSA.

Type	Source	Supported
Message	Formal Documentation	Native
Principal	Formal Documentation	Native
Nonce	Formal Documentation	Native
Timestamp	Extended Literature Review	Non-Native
Symmetric Cryptographic Key	Formal Documentation	Native
Asymmetric Cryptographic Keys	Formal Documentation	Native
Symmetric MAC	Formal Documentation	Native
Text	Formal Documentation	Native
Number	Extended Literature Review	Non-Native

Table 17: CPSA Type Support Criterion Results

b. Operation Support

Table 18 shows the results from the research conducted during the review of CPSA.

Operation	Source	Supported
Symmetric Cryptography	Formal Documentation	Native
Symmetric MAC Generation/Verification	Formal Documentation	Native
Asymmetric Encryption/Decryption	Formal Documentation	Native
Asymmetric Signature/Verification	Formal Documentation	Non-Native
Hashing	Formal Documentation	Native
Addition	Extended Literature Review	Non-Native
Concatenation	Formal Documentation	Native

Table 18: CPSA Operation Support Criterion Results

c. Session Type

From the formal documentation for CPSA it is apparent that CPSA supports infinite sessions.

d. Theoretical Testable Protocol Characteristics

Table 19 shows the results from the research conducted during the review of CPSA.

Testable Protocol Characteristic	Source	Supported
Confidentiality	Formal Documentation	Native
Integrity	Formal Documentation	Unable
Authentication	Formal Documentation	Native
Availability	Formal Documentation	Unable
Non-Repudiation	Formal Documentation	Unable
Round Efficiency	Formal Documentation	Unable

Table 19: CPSA Theoretical Testable Protocol Characteristics Criterion Results

2. Correctness

a. Theoretical Correctness

Table 20 shows the results from the research conducted during the review of CPSA.

Testable Protocol Characteristic	Source	Case
Confidentiality	Formal Documentation	N-I
Integrity	Formal Documentation	Unable
Authentication	Formal Documentation	N-I
Availability	Formal Documentation	Unable
Non-Repudiation	Formal Documentation	Unable
Round Efficiency	Formal Documentation	Unable

Table 20: CPSA Theoretical Correctness Criterion Results

b. Observed Correctness

Table 21 shows the results from the analysis of the data returned from CPSA. The N-S determination is because CPSA failed to halt, given the allotted time, on the Kerberos input.

Testable Protocol Characteristic	Source	Case
Confidentiality	NS Full 2 Implementation NSL Full 2 Implementation Kerberos Implementation	IW-I S-S N-S
Integrity	Formal Documentation	Unable
Authentication	NS Full 2 Implementation NSL Full 2 Implementation Kerberos Implementation	IW-I S-S N-S
Availability	Formal Documentation	Unable
Non-Repudiation	Formal Documentation	Unable
Round Efficiency	Formal Documentation	Unable

Table 21: CPSA Observed Correctness Criterion Results

3. Performance

a. Execution Time

Table 22 summarizes the execution time results collected for CPSA. We note that CPSA failed to halt on the Kerberos input.

Protocol	Execution Time (CI^+ value)
NS	5.49 sec
NSL	13.94 sec
Kerberos	60+ hr

Table 22: CPSA Execution Time Criterion Results

b. Secondary Memory Requirement

The total secondary memory required for CPSA is 8.348 MB.

c. Main Memory Requirement

Table 23 is the summary of the secondary memory requirement results for CPSA.

Protocol	Main Memory Requirement (CI^+ value)
NS	0 KB
NSL	0 KB
Kerberos	Failed to Halt

Table 23: CPSA Main Memory Requirement Criterion Results

4. Usability

a. Automation

The formal documentation for CPSA describes how to use CPSA in both the interactive and batch modes. The ability to operate in a batch mode is evidence that CPSA is automated.

b. Specification Comments

The review of the formal documentation for CPSA shows that flat comments are allowed within protocol specifications. Specifications begin with `(*` and end with `*)`.

5. Overall

Overall CPSA achieved a score of 1133 using our evaluation ratings and weightings.

B. PROVERIF

ProVerif is a protocol analysis system primarily designed by Bruno Blanchet, with support from Matín Abadi and Cédric Fournet [ABAD2003, ABAD2004, ABAD2005, BLAN2001]. ProVerif accepts two types of input grammars: horn clauses, and spi calculus. In practice ProVerif translates spi calculus specifications into horn clauses. We implemented the test protocols in the spi calculus grammar.

1. Scope

a. Type Support

The summary of the type support results for ProVerif are presented in Table 24.

Type	Source	Supported
Message	Formal Documentation	Native
Principal	Formal Documentation	Native
Nonce	Formal Documentation	Native
Timestamp	Formal Documentation	Non-Native
Symmetric Cryptographic Key	Formal Documentation	Native
Asymmetric Cryptographic Keys	Formal Documentation	Native
Symmetric MAC	Formal Documentation	Native
Text	Formal Documentation	Native
Number	Formal Documentation	Non-Native

Table 24: ProVerif Type Support Criterion Results

b. Operation Support

Table 25 presents the summary of the operation support data for ProVerif.

Operation	Source	Supported
Symmetric Cryptography	Formal Documentation	Native
Symmetric MAC Generation/Verification	Formal Documentation	Native
Asymmetric Encryption/Decryption	Formal Documentation	Native
Asymmetric Signature/Verification	Formal Documentation	Native
Hashing	Formal Documentation	Native
Addition	Extended Literature Review	Non-Native
Concatenation	Formal Documentation	Native

Table 25: ProVerif Operation Support Criterion Results

c. Session Type

Through the review of ProVerif's formal documentation it is clear that ProVerif supports infinite session types.

d. Theoretical Testable Protocol Characteristics

Table 26 presents the theoretical testable protocol characteristics for ProVerif.

Testable Protocol Characteristic	Source	Supported
Confidentiality	Formal Documentation	Native
Integrity	Formal Documentation	Unable
Authentication	Formal Documentation	Native
Availability	Formal Documentation	Unable
Non-Repudiation	Formal Documentation	Unable
Round Efficiency	Formal Documentation	Unable

Table 26: ProVerif Theoretical Testable Protocol Characteristics Criterion Results

2. Correctness

a. Theoretical Correctness

Table 27 summarizes the theoretical correctness data for ProVerif.

Testable Protocol Characteristic	Source	Case
Confidentiality	Formal Documentation	N-I
Integrity	Formal Documentation	Unable
Authentication	Formal Documentation	N-I
Availability	Formal Documentation	Unable
Non-Repudiation	Formal Documentation	Unable
Round Efficiency	Formal Documentation	Unable

Table 27: ProVerif Theoretical Correctness Criterion Results

b. Observed Correctness

The observed correctness data for ProVerif is presented in Table 28.

Testable Protocol Characteristic	Source	Case
Confidentiality	NS Full 2 Implementation NSL Full 2 Implementation Kerberos Implementation	IW-I S-S S-S
Integrity	Formal Documentation	Unable
Authentication	NS Full 2 Implementation NSL Full 2 Implementation Kerberos Implementation	IW-I S-S IO-S
Availability	Formal Documentation	Unable
Non-Repudiation	Formal Documentation	Unable
Round Efficiency	Formal Documentation	Unable

Table 28: ProVerif Observed Correctness Criterion Results

3. Performance

a. Execution Time

Table 29 presents the execution time data for ProVerif.

Protocol	Execution Time (CI^+ value)
NS	0.09 sec
NSL	0.05 sec
Kerberos	0.03 sec

Table 29: ProVerif Execution Time Criterion Results

b. Secondary Memory Requirement

The total secondary memory required for ProVerif is 6.124 MB.

c. Main Memory Requirement

Table 30 presents the secondary memory requirement data for ProVerif.

Protocol	Main Memory Requirement (CI^+ value)
NS	0 KB
NSL	0 KB
Kerberos	0 KB

Table 30: ProVerif Main Memory Requirement Criterion Results

4. Usability

a. Automation

The formal documentation for ProVerif describes that it is automated analysis system.

b. Specification Comments

The review of the formal documentation for ProVerif shows that flat comments are allowed within protocol specifications. Specifications begin with `(*` and end with `*)`.

5. Overall

Overall ProVerif achieved a score of 1186 using our evaluation ratings and weightings.

THIS PAGE INTENTIONALLY LEFT BLANK

V. CONCLUSIONS

In this work we developed the first evaluation methodology for protocol analysis systems. We began by discussing why there is a need for an evaluation methodology for protocol analysis systems based on the current state of the field. We then went into deeper discussions about the evaluation methodology itself.

We spoke of the four categories of criteria that comprise the evaluation methodology: scope, correctness, performance, and usability. For each of the categories we defined the criteria that are used to evaluate protocol analysis systems on. We then moved to our discussion of the testing methods. We first described pre-testing actions that needed to occur before criteria testing could begin. We then discussed the testing method for each of the evaluation criterion. We then presented the post-testing actions that needed to occur after the evaluation criteria had been tested. We also presented requirements and recommendations to the ordering of the testing methods. We then switched to the subjective part of the evaluation methodology, the ratings and weightings section. We first presented the rating system for each of the criterion, before discussing the overall weighting function.

With the three main parts of the methodology defined: criteria, methods, and ratings and weightings; we discussed how the "living" nature of the methodology can be realized. We presented processes to adapt the methodology and reasons as to why the methodology should be adapted. One of the key aspects of the methodology is its ability to adapt to the needs of the evaluator. Additionally, the methodology we present has the ability to grow with the protocol analysis field and the passage of time. Over time the results produced through the use of this methodology will benefit the evaluators and the protocol analysis field in general.

Finally, we presented two example evaluations of CPSA and ProVerif, to show that the evaluation methodology could in fact be used and produce data.

A. FUTURE WORK

The next obvious step to take is to complete the definition of the subjective usability testing methods using the process discussed in III.D METHODOLOGY

ADAPTATIONS. Along with the testing methods, the rating systems for each of the subjective usability criteria should be defined. Finally, the overall weighting function can be adapted to incorporate the subjective usability criteria.

Currently we use standard constant definitions of confidentiality, integrity, authentication, and availability. In practice, some protocol analysis systems have different definitions of these terms. A more elaborate rating system may be able to account for these differences across analysis systems.

In the beginning of this work we mentioned that this evaluation methodology should be thought of as a "living" methodology. As the protocol analysis field continues to grow so should this methodology. Additionally, the passage of time will also cause a need to review and update the methodology. We envision that the evaluation methodology will be reviewed and updated about every five to seven years.

Regardless of when the above proposals are implemented the evaluation methodology can still be used "as is". The simplest work that can be done is to evaluate more protocol analysis systems under the current methodology[20]. As mentioned previously, the data from the evaluation methodology can be used by a variety of people. The data from different evaluators can be shared, thus increasing the value of evaluation efforts.

We have mentioned throughout this work that a key aspect of this methodology is its ability to adapt; especially the methodology's ability to adapt to the needs of the evaluator. New weighting functions can be developed to address the needs of different evaluators, such as educators, or professional analysis organizations like the IETF.

[20] In APPENDIX D: we present a list of current protocol analysis systems.

APPENDIX A: TEST PLATFORM CONFIGURATION

In this appendix we present our test platform configuration information.

A. PLATFORM SPECIFICATIONS

Platform Manufacturer:	Dell
Platform:	Optiplex GX270
BIOS Version:	A04
Processor Manufacturer:	Intel
Processor Family:	Pentium 4 (x86)
Processor Speed:	3.0 GHz
Bus Speed:	800 MHz
Level 2 Cache:	1 MB
RAM Type:	DDR SDRAM
RAM Size:	2 GB (4 x 512 MB)
Memory Speed:	333 MHz

Hyper-threading was enabled in the BIOS.

B. FEDORA CONFIGURATION

OS Manufacturer:	Red Hat
OS Version:	Fedora Core 5™ (2.6.18-2200.fc5smp)
Patched as of Date:	15 November 2006
Primary Partition Size:	18 GB
Swap Partition Size:	2 GB

The cups software was not updated due to other system configuration conflicts; the cups version of the system was 1:1.2.5-1.fc5.4.i386, the cups-libs version of the system was 1:1.2.5-1.fc5.4.i386.

1. CPSA Configuration

System Manufacturer:	The MITRE Corporation
System Full Name:	Cryptographic Protocol Shapes Analyzer (CPSA)

System Version: 0.81

The standard installation and configuration as per the formal documentation were used for CPSA.

2. ProVerif Configuration

System Manufacturer: Bruno Blanchet and Martín Abadi

System Full Name: ProVerif

System Version: 1.13pl7

ProVerif requires that Objective Caml (OCaml) be installed on the system, OCaml version 3.09.3 was installed.

C. WINDOWS CONFIGURATION

OS Manufacturer: Microsoft

OS Version: Windows XP SP2

Patched as of Date: 15 November 2006

Primary Partition Size: 20 GB

Virtual Memory Size: 2 GB

APPENDIX B: FILE LISTINGS

Here we present the testing scripts and specifications that were used in this evaluation.

Listing 1 is the master script file that was used to call the other test scripts.

```
#!/bin/bash
# File:  master.sh
# Created By: Chris W. Hoffmeister
# Created On: 25 Jan 2007
# Modified On: -
# Modified By: -
# Description: Master script that runs all test scripts and
appropriately
#              redirects stderr

# Setup variables
proverifdir=/usr/bin/proverif1.13pl7
resultsdir=/share/thesis/data/results
scriptsdir=/share/thesis/data/scripts
specsdir=/share/thesis/data/specifications
exectrials=51

# CPSA test scripts
# Observed Correctness Criteria Scripts
cpsa -b < $specsdir/cpsa/NS_simple_2_cpsa.cpsa >
$resultsdir/cpsa/NS_simple_2_cpsa_corr.txt
cpsa -b < $specsdir/cpsa/NS_full_2_cpsa.cpsa >
$resultsdir/cpsa/NS_full_2_cpsa_corr.txt
cpsa -b < $specsdir/cpsa/NSL_simple_2_cpsa.cpsa >
$resultsdir/cpsa/NSL_simple_2_cpsa_corr.txt
cpsa -b < $specsdir/cpsa/NSL_full_2_cpsa.cpsa >
$resultsdir/cpsa/NSL_full_2_cpsa_corr.txt
# Performance Criteria Scripts
# Secondary Memory Requirement Script
$scriptsdir/./cpsa_second_mem.sh
# Execution Time and Main Memory Requirement Scripts
```

```
$scriptsdir/./NS_simple_2_cpsa.sh $exectrials 2>>
$resultsdir/cpsa/NS_simple_2_cpsa_exec.txt
$scriptsdir/./NS_full_2_cpsa.sh $exectrials 2>>
$resultsdir/cpsa/NS_full_2_cpsa_exec.txt
$scriptsdir/./NSL_simple_2_cpsa.sh $exectrials 2>>
$resultsdir/cpsa/NSL_simple_2_cpsa_exec.txt
$scriptsdir/./NSL_full_2_cpsa.sh $exectrials 2>>
$resultsdir/cpsa/NSL_full_2_cpsa_exec.txt

# ProVerif test scripts
# Observed Correctness Criteria Scripts
$proverifdir/./analyzer $specsdir/proverif/NS_simple_2_proverif_pi.txt
> $resultsdir/proverif/NS_simple_2_proverif_pi_corr.txt
$proverifdir/./analyzer -in pi
$specsdir/proverif/NS_full_2_proverif_pi.txt >
$resultsdir/proverif/NS_full_2_proverif_pi_corr.txt
$proverifdir/./analyzer -in pi
$specsdir/proverif/NSL_simple_2_proverif_pi.txt >
$resultsdir/proverif/NSL_simple_2_proverif_pi_corr.txt
$proverifdir/./analyzer -in pi
$specsdir/proverif/NSL_full_2_proverif_pi.txt >
$resultsdir/proverif/NSL_full_2_proverif_pi_corr.txt
$proverifdir/./analyzer -in pi
$specsdir/proverif/kerberos_proverif_pi.txt >
$resultsdir/proverif/kerberos_proverif_pi_corr.txt
# Performance Criteria Scripts
# Secondary Memory Requirement Script
$scriptsdir/./proverif_second_mem.sh
# Execution Time and Main Memory Requirement Scripts
$scriptsdir/./NS_simple_2_proverif_pi.sh $exectrials 2>>
$resultsdir/proverif/NS_simple_2_proverif_pi_exec.txt
$scriptsdir/./NS_full_2_proverif_pi.sh $exectrials 2>>
$resultsdir/proverif/NS_full_2_proverif_pi_exec.txt
$scriptsdir/./NSL_simple_2_proverif_pi.sh $exectrials 2>>
$resultsdir/proverif/NSL_simple_2_proverif_pi_exec.txt
$scriptsdir/./NSL_full_2_proverif_pi.sh $exectrials 2>>
$resultsdir/proverif/NSL_full_2_proverif_pi_exec.txt
```

```
$scriptsdir/./kerberos_proverif_pi.sh $exectrials 2>>
$resultsdir/proverif/kerberos_proverif_pi_exec.txt
```

Listing 1: Master Test Script

A. CPSA

1. Protocol Specifications

a. NS Simple 2

Listing 2 is the CPSA batch mode specification of the simple version of NS protocol 2 from [NEED1978].

```
(*
 * File:         NS_simple_2_cpsa.cpsa
 * Created By:   Chris W. Hoffmeister
 * Created On:   21 Jan 2007
 * Modified By:  -
 * Modified On:  -
 * Description:  Simplified specification for the Needham-Schroeder
Protocol 2,
 *               from [NEED1978] for the Cryptographic Protocol Shapes
Analyzer
 *               (CPSA) protocol analysis system. This specification is
meant
 *               to be run via the batch mode of CPSA.
 *)

(*
 * Protocol Specification
 *)
protocol ns_simple_2=(

(*
 * Initiator Role, Alice
 *)
    role init(A:name; B:name; Na:nonce; Nb:nonce)=(
        non=(privk(A))
        uniq=(Na)
```

```
        messages=(
            + {A, Na}pubk(B);
            - {Na, Nb}pubk(A);
            + {Nb}pubk(B);
        );
    )

(*
 * Responder Role, Bob
 *)
    role resp(A:name; B:name; Na:nonce; Nb:nonce)=(
        non=(privk(B))
        uniq=(Nb)
        messages=(
            - {A, Na}pubk(B);
            + {Na, Nb}pubk(A);
            - {Nb}pubk(B);
        )
    )
)

(*
 * Analyzer Commands
 *)
(*
 * Test the initiator role, Alice
 *)
create init[3]
non privk(B) (* Assume the single instance of the responder is honest
*)
printskel
discover
(*
 * Test the responder role, Bob
 *)
create resp[3]
```

```
non privk(A) (* Assume the single instance of the initiator is honest
*)
printskel
discover
```

Listing 2: CPSA Batch Mode NS Simple 2 Protocol Specification

b. NS Full 2

Listing 3 is the NS Full 2 protocol specification for the batch mode of CPSA [NEED1978].

```
(*
 * File:         NS_full_2_cpsa.cpsa
 * Created By:   Chris W. Hoffmeister
 * Created On:   21 Jan 2007
 * Modified By:  -
 * Modified On:  -
 * Description:  Full specification for the Needham-Schroeder Protocol
2, from
 *               [NEED1978] for the Cryptographic Protocol Shapes
Analyzer
 *               (CPSA) protocol analysis system. This specification is
meant
 *               to be run via the batch mode of CPSA.
 *)

(*
 * Protocol Specification
 *)
protocol ns_full_2=(

(*
 * Initiator Role, Alice
 *)
    role init(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
        non=(privk(A))
        uniq=(Na)
        messages=(
            + (A, B);
```

```
            - {B, pubk(B)}privk(S);
            + {A, Na}pubk(B);
            - {Na, Nb}pubk(A);
            + {Nb}pubk(B);
        )
    )

(*
 * Responder Role, Bob
 *)
    role resp(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
        non=(privk(B))
        uniq=(Nb)
        messages=(
            - {A, Na}pubk(B);
            + (B, A);
            - {A, pubk(A)}privk(S);
            + {Na, Nb}pubk(A);
            - {Nb}pubk(B);
        )
    )

(*
 * Server Role, Server
 *)
    role serv(A:name; B:name; S:name)=(
        non=(privk(S))
        uniq=()
        messages=(
            - (A, B);
            + {B, pubk(B)}privk(S);
            - (B, A);
            + {A, pubk(A)}privk(S);
        )
    )
)
```

```
(*
 * Analyzer Commands
 *)
(*
 * Test the initiator role, Alice
 *)
create init[5]
non privk(B) (* Assume the single instance of the responder is honest
*)
printskel
discover
(*
 * Test the responder role, Bob
 *)
create resp[5]
non privk(A) (* Assume the single instance of the initiator is honest
*)
printskel
discover
```

Listing 3: CPSA Batch Mode NS Full 2 Protocol Specification

c. NSL Simple 2

Listing 4 is the batch mode specification of the NSL Simple 2 protocol for CPSA [LOWE1996].

```
(*
 * File:         NSL_simple_2_cpsa.cpsa
 * Created By:   Chris W. Hoffmeister
 * Created On:   21 Jan 2007
 * Modified By:  -
 * Modified On:  -
 * Description:  Simplified specification for the Needham-Schroeder-
Lowe
 *               Protocol from [LOWE1996] for the Cryptographic
Protocol
 *               Shapes Analyzer (CPSA) protocol analysis system. This
 *               specification is meant to be run via the batch mode of
CPSA.
```

```
 *)

(*
 * Protocol Specification
 *)
protocol nsl_simple_2=(

(*
 * Initiator Role, Alice
 *)
    role init(A:name; B:name; Na:nonce; Nb:nonce)=(
        non=(privk(A))
        uniq=(Na)
        messages=(
            + {A, Na}pubk(B);
            - {B, Na, Nb}pubk(A);
            + {Nb}pubk(B);
        );
    )

(*
 * Responder Role, Bob
 *)
    role resp(A:name; B:name; Na:nonce; Nb:nonce)=(
        non=(privk(B))
        uniq=(Nb)
        messages=(
            - {A, Na}pubk(B);
            + {B, Na, Nb}pubk(A);
            - {Nb}pubk(B);
        )
    )
)

(*
 * Analyzer Commands
 *)
```

```
(*
 * Test the initiator role, Alice
 *)
create init[3]
non privk(B) (* Assume the single instance of the responder is honest
*)
printskel
discover
(*
 * Test the responder role, Bob
 *)
create resp[3]
non privk(A) (* Assume the single instance of the initiator is honest
*)
printskel
discover
```

Listing 4: CPSA Batch Mode NSL Simple 2 Protocol Specification

d. NSL Full 2

Listing 5 is the batch mode specification of the NSL Full 2 protocol for CPSA [LOWE1996].

```
(*
 * File:         NSL_full_2_cpsa.cpsa
 * Created By:   Chris W. Hoffmeister
 * Created On:   21 Jan 2007
 * Modified By:  -
 * Modified On:  -
 * Description:  Full specification for the Needham-Schroeder-Lowe
Protocol
 *               from [LOWE1996] for the Cryptographic Protocol Shapes
Analyzer
 *               (CPSA) protocol analysis system. This specification is
meant
 *               to be run via the batch mode of CPSA.
 *)

(*
```

```
 * Protocol Specification
 *)
protocol nsl_full_2=(

(*
 * Initiator Role, Alice
 *)
    role init(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
        non=(privk(A))
        uniq=(Na)
        messages=(
            + (A, B);
            - {B, pubk(B)}privk(S);
            + {A, Na}pubk(B);
            - {B, Na, Nb}pubk(A);
            + {Nb}pubk(B);
        )
    )

(*
 * Responder Role, Bob
 *)
    role resp(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
        non=(privk(B))
        uniq=(Nb)
        messages=(
            - {A, Na}pubk(B);
            + (B, A);
            - {A, pubk(A)}privk(S);
            + {B, Na, Nb}pubk(A);
            - {Nb}pubk(B);
        )
    )

(*
 * Server Role, Server
 *)
```

```
    role serv(A:name; B:name; S:name)=(
        non=(privk(S))
        uniq=()
        messages=(
            - (A, B);
            + {B, pubk(B)}privk(S);
            - (B, A);
            + {A, pubk(A)}privk(S);
        )
    )
)

(*
 * Analyzer Commands
 *)
(*
 * Test the initiator role, Alice
 *)
create init[5]
non privk(B) (* Assume the single instance of the responder is honest
*)
printskel
discover
(*
 * Test the responder role, Bob
 *)
create resp[5]
non privk(A) (* Assume the single instance of the initiator is honest
*)
printskel
discover
```

Listing 5: CPSA Batch Mode NSL Full 2 Protocol Specification

e. Kerberos

Listing 6 is the batch mode specification of the Kerberos protocol for CPSA [RFC4120].

```
(*
```

```
 * File:          kerberos_cpsa.cpsa
 * Created By:    Chris W. Hoffmeister
 * Created On:    21 Jan 2007
 * Modified By:   -
 * Modified On:   -
 * Description:   Specification for the Kerberos V5 Protocol from RFC
4120 of
 *                July 2005 for the Cryptographic Protocol Shapes
Analyzer
 *                (CPSA) protocol analysis system. This specification is
meant
 *                to be run via the batch mode of CPSA.
 *)

(*
 * Protocol Specification
 *)
protocol kerberos=(

(*
 * User Role, Alice
 *)
    role user(A:name; B:name; KAS:name; TGS:name; Nak:nonce; Nat:nonce;
              Tk:nonce; Tat:nonce; Tt:nonce; Tab:nonce; Kat:sessk;
Kab:sessk)=(
        non=(ltk(A;KAS))
        uniq=(Nak; Nat; Tat; Tab)
        messages=(
            + A, TGS, Nak;
            - A, {A, Kat, Tk}ltk(KAS;TGS), {TGS, Kat, Nak,
Tk}ltk(A;KAS);
            + B, Nat, {A, Kat, Tk}ltk(KAS;TGS), {A, Tat}Kat;
            - A, {A, Kab, Tt}ltk(B;TGS), {B, Kab, Nat, Tt}Kat;
            + {A, Kab, Tt}ltk(B;TGS), {A, Tab}Kab;
            - {Tab}Kab;
        )
    )
```

```
(*
 * Application Server Role, Bob
 *)
    role appserv(A:name; B:name; TGS:name; Tt:nonce; Tab:nonce;
Kab:sessk)=(
        non=(ltk(B;TGS))
        uniq=()
        messages=(
            - {A, Kab, Tt}ltk(B;TGS), {A, Tab}Kab;
            + {Tab}Kab;
        )
    )

(*
 * Authentication Server Role
 *)
    role authserv(A:name; KAS:name; TGS:name; Nak:nonce; Tk:nonce;
Kat:sessk)=(
        non=(ltk(A;KAS); ltk(KAS;TGS))
        uniq=(Tk; Kat)
        messages=(
            - A, TGS, Nak;
            + A, {A, Kat, Tk}ltk(KAS;TGS), {TGS, Kat, Nak,
Tk}ltk(A;KAS);
        )
    )

(*
 * Ticket Distribution Server Role
 *)
    role ticserv(A:name; B:name; KAS:name; TGS:name; Nat:nonce;
Tk:nonce;
                 Tat:nonce; Tt:nonce; Kat:sessk; Kab:sessk)=(
        non=(ltk(KAS;TGS); ltk(B;TGS))
        uniq=(Tt; Kab)
        messages=(
```

```
            - B, Nat, {A, Kat, Tk}ltk(KAS;TGS), {A, Tat}Kat;
            + A, {A, Kab, Tt}ltk(B;TGS), {B, Kab, Nat, Tt}Kat;
        )
    )

)

(*
 * Analyzer Commands
 *)
create user[6]
printskel
discover
create appserv[1]
printskel
discover
```

Listing 6: CPSA Batch Mode Kerberos Protocol Specification

2. Performance Criteria Test Scripts

a. Execution Time & Main Memory Requirement

Listing 7 is the test script to run the execution time and main memory requirements tests for CPSA for the NS simple 2 protocol.

```
#!/bin/bash
# File:        NS_simple_2_cpsa.sh
# Created By:  Chris W. Hoffmeister
# Created On:  25 Jan 2007
# Modified By: -
# Modified On: -
# Description: Shell script to test Execution Time of
NS_simple_2_cpsa.cpsa

# Setup variables
resultsdir=/share/thesis/data/results
specsdir=/share/thesis/data/specifications
trials=5 # Default number of experiments to run
```

```
# Determines if the number of trials was passed in
if [ $# -eq 1 ]
then
# Number of trials was passed in
    trials=$1
fi

echo "Begin Test: Execution Time: CPSA-NS Simple 2" >
$resultsdir/cpsa/NS_simple_2_cpsa_exec.txt
date >> $resultsdir/cpsa/NS_simple_2_cpsa_exec.txt

# Perform trials
for ((trial=0; trial<trials; trial=trial+1)) do
    echo "Trial:  $trial" >> $resultsdir/cpsa/NS_simple_2_cpsa_exec.txt
    # We're not concerned with the output, merely the execution time
    # Have to explicitly map to time function, other wise
/bin/bash/time will be used
    /usr/bin/time -f "Command:        %C\nUser Time:
%U(sec)\nSystem Time:    %S(sec)\nMaximum Memory: %M(KB)\nExit Status:
%x" cpsa -b < $specsdir/cpsa/NS_simple_2_cpsa.cpsa > /dev/null
done

echo "End Test:   Execution Time: CPSA-NS Simple 2" >>
$resultsdir/cpsa/NS_simple_2_cpsa_exec.txt
```

Listing 7: CPSA Batch Mode NS Simple 2 Protocol Test Script

Listing 8 is the NS Full 2 protocol execution time and main memory requirement test script for CPSA.

```
#!/bin/bash
# File:        NS_full_2_cpsa.sh
# Created By:  Chris W. Hoffmeister
# Created On:  25 Jan 2007
# Modified By: -
# Modified On: -
# Description: Shell script to test Execution Time of
NS_full_2_cpsa.cpsa
```

```
# Setup variables
resultsdir=/share/thesis/data/results
specsdir=/share/thesis/data/specifications
trials=5 # Default number of experiments to run

# Determines if the number of trials was passed in
if [ $# -eq 1 ]
then
# Number of trials was passed in
    trials=$1
fi

echo "Begin Test: Execution Time: CPSA-NS Full 2" >
$resultsdir/cpsa/NS_full_2_cpsa_exec.txt
date >> $resultsdir/cpsa/NS_full_2_cpsa_exec.txt

# Perform trials
for ((trial=0; trial<trials; trial=trial+1)) do
    echo "Trial:  $trial" >> $resultsdir/cpsa/NS_full_2_cpsa_exec.txt
    # We're not concerned with the output, merely the execution time
    # Have to explicitly map to time function, other wise
/bin/bash/time will be used
    /usr/bin/time -f "Command:        %C\nUser Time:
%U(sec)\nSystem Time:    %S(sec)\nMaximum Memory: %M(KB)\nExit Status:
%x" cpsa -b < $specsdir/cpsa/NS_full_2_cpsa.cpsa > /dev/null
done

echo "End Test:   Execution Time: CPSA-NS Full 2" >>
$resultsdir/cpsa/NS_full_2_cpsa_exec.txt
```

Listing 8: CPSA Batch Mode NS Full 2 Protocol Test Script

Listing 9 is the NSL Simple 2 protocol execution time and main memory requirement test script for CPSA.

```
#!/bin/bash
# File:        NSL_simple_2_cpsa.sh
# Created By:  Chris W. Hoffmeister
# Created On:  25 Jan 2007
```

```
# Modified By: -
# Modified On: -
# Description: Shell script to test Execution Time of
NSL_simple_2_cpsa.cpsa

# Setup variables
resultsdir=/share/thesis/data/results
specsdir=/share/thesis/data/specifications
trials=5 # Default number of experiments to run

# Determines if the number of trials was passed in
if [ $# -eq 1 ]
then
# Number of trials was passed in
    trials=$1
fi

echo "Begin Test: Execution Time: CPSA-NSL Simple 2" >
$resultsdir/cpsa/NSL_simple_2_cpsa_exec.txt
date >> $resultsdir/cpsa/NSL_simple_2_cpsa_exec.txt

# Perform trials
for ((trial=0; trial<trials; trial=trial+1)) do
    echo "Trial:  $trial" >>
$resultsdir/cpsa/NSL_simple_2_cpsa_exec.txt
    # We're not concerned with the output, merely the execution time
    # Have to explicitly map to time function, other wise
/bin/bash/time will be used
    /usr/bin/time -f "Command:        %C\nUser Time:
%U(sec)\nSystem Time:    %S(sec)\nMaximum Memory: %M(KB)\nExit Status:
%x" cpsa -b < $specsdir/cpsa/NSL_simple_2_cpsa.cpsa > /dev/null
done

echo "End Test:   Execution Time: CPSA-NSL Simple 2" >>
$resultsdir/cpsa/NSL_simple_2_cpsa_exec.txt
```

Listing 9: CPSA Batch Mode NSL Simple 2 Protocol Test Script

Listing 10 is the NSL Full 2 protocol execution time and main memory requirement test script for CPSA.

```
#!/bin/bash
# File:        NSL_full_2_cpsa.sh
# Created By:  Chris W. Hoffmeister
# Created On:  25 Jan 2007
# Modified By: -
# Modified On: -
# Description: Shell script to test Execution Time of
NSL_full_2_cpsa.cpsa

# Setup variables
resultsdir=/share/thesis/data/results
specsdir=/share/thesis/data/specifications
trials=5 # Default number of experiments to run

# Determines if the number of trials was passed in
if [ $# -eq 1 ]
then
# Number of trials was passed in
    trials=$1
fi

echo "Begin Test: Execution Time: CPSA-NSL Full 2" >
$resultsdir/cpsa/NSL_full_2_cpsa_exec.txt
date >> $resultsdir/cpsa/NSL_full_2_cpsa_exec.txt

# Perform trials
for ((trial=0; trial<trials; trial=trial+1)) do
    echo "Trial:  $trial" >> $resultsdir/cpsa/NSL_full_2_cpsa_exec.txt
    # We're not concerned with the output, merely the execution time
    # Have to explicitly map to time function, other wise
/bin/bash/time will be used
    /usr/bin/time -f "Command:        %C\nUser Time:
%U(sec)\nSystem Time:    %S(sec)\nMaximum Memory: %M(KB)\nExit Status:
%x" cpsa -b < $specsdir/cpsa/NSL_full_2_cpsa.cpsa > /dev/null
done
```

```
echo "End Test:   Execution Time: CPSA-NSL Full 2" >>
$resultsdir/cpsa/NSL_full_2_cpsa_exec.txt
```

Listing 10: CPSA Batch Mode NSL Full 2 Protocol Test Script

b. Secondary Memory Requirement

Listing 11 is the secondary memory requirement test script for CPSA.

```
#!/bin/bash
# File:  cpsa_second_mem.sh
# Created By: Chris W. Hoffmeister
# Created On: 06 Feb 2007
# Modified On: -
# Modified By: -
# Description: Script that runs ls to determine CPSA size on disk

# Setup variables
tooldir=/usr/local/cpsa-0.81
resultsdir=/share/thesis/data/results

# Perform experiment
ls -lkR $tooldir | grep -F "$tooldir
total " > $resultsdir/cpsa/cpsa_second_mem.txt
```

Listing 11: CPSA Secondary Memory Requirement Test Script

B. PROVERIF

1. Protocol Specifications

a. NS Full 2

Listing 12 is the spi calculus NS Full 2 protocol specification for ProVerif.

```
(************************************************************
 *                                                          *
 *       Cryptographic protocol verifier                    *
 *                                                          *
 *       Bruno Blanchet and Xavier Allamigeon               *
 *                                                          *
 *       Copyright (C) INRIA, LIENS, MPII 2000-2006         *
```

```
 *                                                               *
 ***************************************************************)

(*

    This program is free software; you can redistribute it and/or
modify
    it under the terms of the GNU General Public License as published
by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details (in file LICENSE).

    You should have received a copy of the GNU General Public License
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-
1307  USA

*)
free c.
(*

Needham-Schroeder public key protocol

Message 1: A -> S : (A, B)
Message 2: S -> A : { pkB, B }skS
Message 3: A -> B : { Na, A }pkB
Message 4: B -> S : (B, A)
Message 5: S -> B : { pkA, A }skS
Message 6: B -> A : { Na, Nb }pkA
Message 7: A -> B : { Nb }pkB

The heart of the protocol is messages 3, 6, 7.
```

```
*)

(* Public key cryptography *)

fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),y) = x.

(* Host names
   The server has a table (host name, public key), which we
   represent by the function getkey. *)

fun host/1.
private reduc getkey(host(x)) = x.

(* Signatures *)

fun sign/2.
reduc checksign(sign(x,y),pk(y)) = x.
reduc getmess(sign(x,y)) = x.

(* Shared-key cryptography *)

fun sencrypt/2.
reduc sdecrypt(sencrypt(x,y),y) = x.

(* Secrecy assumptions *)

not skA.
not skB.
not skS.

private free secretANa, secretANb, secretBNa, secretBNb.
query attacker:secretANa;
      attacker:secretANb;
      attacker:secretBNa;
```

```
      attacker:secretBNb.
query ev:endBparam(x) ==> ev:beginBparam(x).
query ev:endBfull(x1,x2,x3,x4,x5,x6) ==>
ev:beginBfull(x1,x2,x3,x4,x5,x6).
query ev:endAparam(x) ==> ev:beginAparam(x).
query ev:endAfull(x1,x2,x3,x4,x5,x6) ==>
ev:beginAfull(x1,x2,x3,x4,x5,x6).
query evinj:endBparam(x) ==> evinj:beginBparam(x).
query evinj:endBfull(x1,x2,x3,x4,x5,x6) ==>
evinj:beginBfull(x1,x2,x3,x4,x5,x6).
query evinj:endAparam(x) ==> evinj:beginAparam(x).
query evinj:endAfull(x1,x2,x3,x4,x5,x6) ==>
evinj:beginAfull(x1,x2,x3,x4,x5,x6).

let processA =
      (* Choose the other host *)
      in(c,hostX);
      event beginBparam(hostX);
      (* Message 1: Get the public key certificate for the other host
*)
      out(c, (hostA, hostX));
      (* Messafe 2 *)
      in(c, ms);
      let (pkX,=hostX) = checksign(ms,pkS) in
        (* Message 3 *)
      new Na;
        out(c, encrypt((Na, hostA), pkX));
        (* Message 6 *)
        in(c, m);
      let (=Na, NX2) = decrypt(m, skA) in
      event beginBfull(Na, hostA, hostX, pkX, pkA, NX2);
        (* Message 7 *)
        out(c, encrypt(NX2, pkX));
        (* OK *)
      if hostX = hostB then
      event endAparam(hostA);
      event endAfull(Na, hostA, hostX, pkX, pkA, NX2);
```

```
      out(c, sencrypt(secretANa, Na));
      out(c, sencrypt(secretANb, NX2)).

let processB =
        (* Message 3 *)
      in(c, m);
      let (NY, hostY) = decrypt(m, skB) in
      event beginAparam(hostY);
      (* Message 4: Get the public key certificate for the other host
*)
        out(c, (hostB, hostY));
      (* Message 5 *)
      in(c,ms);
        let (pkY,=hostY) = checksign(ms,pkS) in
        (* Message 6 *)
      new Nb;
      event beginAfull(NY, hostY, hostB, pkB, pkY, Nb);
      out(c, encrypt((NY, Nb), pkY));
        (* Message 7 *)
      in(c, m3);
        if Nb = decrypt(m3, skB) then
      (* OK *)
        if hostY = hostA then
      event endBparam(hostB);
      event endBfull(NY, hostY, hostB, pkB, pkA, Nb);
      out(c, sencrypt(secretBNa, NY));
      out(c, sencrypt(secretBNb, Nb)).

let processS =  in(c,m);
              let (a,b) = m in
            let sb = getkey(b) in
                out(c,sign((sb,b),skS)).

process new skA; let pkA = pk(skA) in
        out(c, pkA);
        new skB; let pkB = pk(skB) in
        out(c, pkB);
```

```
new skS; let pkS = pk(skS) in
out(c, pkS);
let hostA = host(pkA) in
out(c, hostA);
let hostB = host(pkB) in
out(c, hostB);
((!processA) | (!processB) | (!processS))
```

Listing 12: ProVerif Spi Calculus NS Full 2 Protocol Specification

b. NSL Full 2

Listing 13 is the spi calculus NSL Full 2 protocol specification for ProVerif.

```
(************************************************************
 *                                                          *
 *       Cryptographic protocol verifier                    *
 *                                                          *
 *       Bruno Blanchet and Xavier Allamigeon               *
 *                                                          *
 *       Copyright (C) INRIA, LIENS, MPII 2000-2006         *
 *                                                          *
 ************************************************************)

(*

    This program is free software; you can redistribute it and/or
modify
    it under the terms of the GNU General Public License as published
by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details (in file LICENSE).

    You should have received a copy of the GNU General Public License
```

```
    along with this program; if not, write to the Free Software
    Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-
1307  USA

*)
free c.
(*

Needham-Schroeder public key protocol
Corrected version by Lowe

Message 1: A -> S : (A, B)
Message 2: S -> A : { B, pkB }_skS
Message 3: A -> B : { N_A, A }_pkB
Message 4: B -> S : (B, A)
Message 5: S -> B : { A, pkA }_skS
Message 6: B -> A : { N_A, N_B, B }_pkA
Message 7: A -> B : { N_B }_pkB

The heart of the protocol is messages 3, 6, 7.

*)

(* Public key cryptography *)

fun pk/1.
fun encrypt/2.
reduc decrypt(encrypt(x,pk(y)),y) = x.

(* Host names
   The server has a table (host name, public key), which we
   represent by the function getkey. *)

fun host/1.
private reduc getkey(host(x)) = x.

(* Signatures *)
```

```
fun sign/2.
reduc checksign(sign(x,y),pk(y)) = x.
reduc getmess(sign(x,y)) = x.

(* Shared-key cryptography *)

fun sencrypt/2.
reduc sdecrypt(sencrypt(x,y),y) = x.

(* Secrecy assumptions *)

not skA.
not skB.
not skS.

private free secretANa, secretANb, secretBNa, secretBNb.
query attacker:secretANa;
      attacker:secretANb;
      attacker:secretBNa;
      attacker:secretBNb.
query ev:endBparam(x) ==> ev:beginBparam(x).
query ev:endBfull(x1,x2,x3,x4,x5,x6) ==>
ev:beginBfull(x1,x2,x3,x4,x5,x6).
query ev:endAparam(x) ==> ev:beginAparam(x).
query ev:endAfull(x1,x2,x3,x4,x5,x6) ==>
ev:beginAfull(x1,x2,x3,x4,x5,x6).
query evinj:endBparam(x) ==> evinj:beginBparam(x).
query evinj:endBfull(x1,x2,x3,x4,x5,x6) ==>
evinj:beginBfull(x1,x2,x3,x4,x5,x6).
query evinj:endAparam(x) ==> evinj:beginAparam(x).
query evinj:endAfull(x1,x2,x3,x4,x5,x6) ==>
evinj:beginAfull(x1,x2,x3,x4,x5,x6).

let processA =
      (* Choose the other host *)
      in(c,hostX);
```

```
      event beginBparam(hostX);
      (* Message 1: Get the public key certificate for the other host
*)
      out(c, (hostA, hostX));
      (* Message 2 *)
      in(c, ms);
      let (pkX,=hostX) = checksign(ms,pkS) in
        (* Message 3 *)
      new Na;
        out(c, encrypt((Na, hostA), pkX));
      (* Message 6 *)
        in(c, m);
        let (=Na, NX2, =hostX) = decrypt(m, skA) in
      event beginBfull(Na, hostA, hostX, pkX, pkA, NX2);
      (* Message 7 *)
        out(c, encrypt(NX2, pkX));
        (* OK *)
      if hostX = hostB then
      event endAparam(hostA);
      event endAfull(Na, hostA, hostX, pkX, pkA, NX2);
      out(c, sencrypt(secretANa, Na));
      out(c, sencrypt(secretANb, NX2)).

let processB =
        (* Message 3 *)
      in(c, m);
      let (NY, hostY) = decrypt(m, skB) in
      event beginAparam(hostY);
      (* Message 4: Get the public key certificate for the other host
*)
        out(c, (hostB, hostY));
      (* Message 5 *)
      in(c,ms);
        let (pkY,=hostY) = checksign(ms,pkS) in
        (* Message 6 *)
      new Nb;
      event beginAfull(NY, hostY, hostB, pkB, pkY, Nb);
```

```
      out(c, encrypt((NY, Nb, hostB), pkY));
        (* Message 7 *)
      in(c, m3);
        if Nb = decrypt(m3, skB) then
      (* OK *)
        if hostY = hostA then
      event endBparam(hostB);
      event endBfull(NY, hostY, hostB, pkB, pkA, Nb);
      out(c, sencrypt(secretBNa, NY));
      out(c, sencrypt(secretBNb, Nb)).

let processS =  in(c,m);
              let (a,b) = m in
            let sb = getkey(b) in
                out(c,sign((sb,b),skS)).

process new skA; let pkA = pk(skA) in
        out(c, pkA);
        new skB; let pkB = pk(skB) in
        out(c, pkB);
      new skS; let pkS = pk(skS) in
      out(c, pkS);
      let hostA = host(pkA) in
      out(c, hostA);
      let hostB = host(pkB) in
      out(c, hostB);
      ((!processA) | (!processB) | (!processS))
```

Listing 13: ProVerif Spi Calculus NSL Full 2 Protocol Specification

c. Kerberos

Listing 14 is the spi calculus Kerberos protocol specification for ProVerif.

```
(*
 * File:    kerberos_proverif_pi.txt
 * Created By:    Chris W. Hoffmeister
 * Created On:    08 Mar 2007
 * Modified By:   -
 * Modified On: -
```

```
 * Description: Specification for the Kerberos V5 Protocol from RFC
4120 of
 *              July 2005 for ProVerif using pi calculus.
 *)

(*
 * Environment setup
 *)
param attacker = active.
param keyCompromise = none.
param traceDisplay = long.

(*
 * These are public, i.e., everyone can use them
 *)
free chnl.
free UserA.
free AppServX.
free TicServA.

(*
 * Definitions, taken from examples provided with ProVerif
 *)

(*
 *Shared-key cryptography
 *)
fun ShareEnc/2.
reduc ShareDec(ShareEnc(Msg,SharedKey),SharedKey) = Msg.

(*
 * Secrecy assupmtions
 *)
not ltkAKAS. (*Long-term key between User (Alice) and Authentication
Server*)
not ltkKASTGS. (*Long-term key between Auth Server and Ticket Server*)
```

```
not ltkAPSTGS. (*Long-term key between Application Server and Ticket
Server*)

(*
 * Queries
 *)
private free AttackSuccess. (*Label for Confidentiality Test*)
query attacker:AttackSuccess. (*Confindentiality Test*)
(*Authtenication Tests*)
query evinj:ProtocolCompleteUser(User, Kuaps) ==>
  evinj:ProtocolTicServ2(User, Kuaps) & evinj:ProtocolAuthServ(User,
Katgs).
query evinj:ProtocolTicServ1(User, Kutgs) ==>
  evinj:ProtocolAuthServ(User, Kutgs).
query evinj:ProtocolAppServ(User, Kuaps) ==>
  evinj:ProtocolTicServ2(User, Kuaps).

(*
 * Protocol specification
 *)

(*
 * User Role, Alice
 *)
let PrcsUser = (
    new Nak; (*Nonce between User and Authentication Server*)
    out(chnl, (UserA, TicServA, Nak)); (*Step 1: User -> AuthServ*)
    in(chnl, (UserX, AuthServTicServ, AuthServUser));
      (*Step 2: AuthServ -> User*)
    let (=UserA) = UserX in (
        let (=TicServA, Kat, =Nak, Tk) = ShareDec(AuthServUser,
ltkAKAS) in (
            new Nat; (*Nonce between User and Ticket Server*)
            new Tat; (*New timestamp from User*)
            out(chnl, (AppServX, Nat, AuthServTicServ,
              ShareEnc((UserA, Tat), Kat))); (*Step 3: User ->
TicServ*)
```

```
            in(chnl, (=UserA, TicServAppServ, TicServUser));
              (*Step 4: TicServ -> User*)
            let (=AppServX, Kab, =Nat, Tt) = ShareDec(TicServUser, Kat)
in (
                new Tab; (*New timestamp form User*)
                out(chnl, (TicServAppServ, ShareEnc((UserA, Tab),
Kab)));
                  (*Step 5: User -> AppServ*)
                in(chnl, AppServUser); (*Step 6: AppServ -> User*)
                let (=Tab) = ShareDec(AppServUser, Kab) in (
                    event ProtocolCompleteUser(UserA, Kab);
                    out(chnl, ShareEnc(AttackSuccess, Kab))
                )
            )
        )
    )
).

(*
 * Application Server Role, Bob
 *)
let PrcsAppServ = (
    in(chnl, (TicServAppServ, UserAppServ)); (*Step 5: User ->
AppServ*)
    let (UserX, Kxaps, Tx) = ShareDec(TicServAppServ, ltkAPSTGS) in (
        let (=UserX, Txaps) = ShareDec(UserAppServ, Kxaps) in (
            out(chnl, ShareEnc(Txaps, Kxaps)); (*Step 6: AppServ ->
User*)
            event ProtocolAppServ(UserX, Kxaps)
        )
    )
).

(*
 * Authentication Server Role
 *)
let PrcsAuthServ = (
```

```
    in(chnl, UserAuthServ); (*Step 1: User -> AuthServ*)
    let (UserX, TicServX, Nxk) = UserAuthServ in (
        new Kxt; (*New shared key between User and Ticket Distribution
Server*)
        new Tk; (*New timestamp from Authentication Server*)
        out(chnl, (UserX, ShareEnc((UserX, Kxt, Tk), ltkKASTGS),
          ShareEnc((TicServX, Kxt, Nxk, Tk), ltkAKAS)));
          (*Step 2: AuthServ -> User*)
        event ProtocolAuthServ(UserX, Kxt)
    )
).

(*
 * Ticket Distribution Server Role
 *)
let PrcsTicServ = (
    in(chnl, (AppServX, Nxt, AuthServTicServ, UserTicServ));
      (*Step 3: User -> TicServ*)
    let (UserX, Kxt, Tk) = ShareDec(AuthServTicServ, ltkKASTGS) in (
        let (=UserX, Txt) = ShareDec(UserTicServ, Kxt) in (
            event ProtocolTicServ1(UserX, Kxt);
            new Kxaps; (*New shared key between User and Application
Server*)
            new Tt; (*New timestamp from Ticket Server*)
            out(chnl, (UserX, ShareEnc((UserX, Kxaps, Tt), ltkAPSTGS),
              ShareEnc((AppServX, Kxaps, Nxt, Tt), Kxt)));
              (*Step 4: TicServ -> User*)
            event ProtocolTicServ2(UserX, Kxaps)
        )
    )
).

(*
 * Protocol Instantiation
 *)
process
    new ltkAKAS;
```

```
    new ltkKASTGS;
    new ltkAPSTGS;
    ((!PrcsUser) | (!PrcsAuthServ) | (!PrcsTicServ) | (!PrcsAppServ))
```

Listing 14: ProVerif Spi Calculus Kerberos Protocol Specification

2. Performance Test Scripts

a. Execution Time & Main Memory Requirement

Listing 15 is the execution time and main memory requirement for ProVerif for the NS Full 2 protocol.

```
#!/bin/bash
# File:        NS_full_2_proverif_pi.sh
# Created By:  Chris W. Hoffmeister
# Created On:  05 Mar 2007
# Modified By: -
# Modified On: -
# Description: Shell script to test Execution Time of
NS_full_2_proverif_pi.txt

# Setup variables
proverifdir=/usr/bin/proverif1.13pl7
resultsdir=/share/thesis/data/results
specsdir=/share/thesis/data/specifications
trials=5 # Default number of experiments to run

# Determines if the number of trials was passed in
if [ $# -eq 1 ]
then
# Number of trials was passed in
    trials=$1
fi

echo "Begin Test: Execution Time: Proverif-NS Full 2" >
$resultsdir/proverif/NS_full_2_proverif_pi_exec.txt
date >> $resultsdir/proverif/NS_full_2_proverif_pi_exec.txt

# Perform trials
```

```
for ((trial=0; trial<trials; trial=trial+1)) do
    echo "Trial:  $trial" >>
$resultsdir/proverif/NS_full_2_proverif_pi_exec.txt
    # We're not concerned with the output, merely the execution time
    # Have to explicitly map to time function, other wise
/bin/bash/time will be used
    /usr/bin/time -f "Command:        %C\nUser Time:
%U(sec)\nSystem Time:    %S(sec)\nMaximum Memory: %M(KB)\nExit Status:
%x" $proverifdir/./analyzer -in pi
$specsdir/proverif/NS_full_2_proverif_pi.txt > /dev/null
done

echo "End Test:   Execution Time: Proverif-NS Full 2" >>
$resultsdir/proverif/NS_full_2_proverif_pi_exec.txt
```

Listing 15: ProVerif Spi Calculus NS Full 2 Protocol Test Script

Listing 16 is the execution time and main memory requirement test script for ProVerif for the NSL Full 2 protocol.

```
#!/bin/bash
# File:        NSL_full_2_proverif_pi.sh
# Created By:  Chris W. Hoffmeister
# Created On:  05 Mar 2007
# Modified By: -
# Modified On: -
# Description: Shell script to test Execution Time of
NSL_full_2_proverif_pi.txt

# Setup variables
proverifdir=/usr/bin/proverif1.13pl7
resultsdir=/share/thesis/data/results
specsdir=/share/thesis/data/specifications
trials=5 # Default number of experiments to run

# Determines if the number of trials was passed in
if [ $# -eq 1 ]
then
# Number of trials was passed in
```

```
    trials=$1
fi

echo "Begin Test: Execution Time: Proverif-NSL Full 2" >
$resultsdir/proverif/NSL_full_2_proverif_pi_exec.txt
date >> $resultsdir/proverif/NSL_full_2_proverif_pi_exec.txt

# Perform trials
for ((trial=0; trial<trials; trial=trial+1)) do
    echo "Trial:  $trial" >>
$resultsdir/proverif/NSL_full_2_proverif_pi_exec.txt
    # We're not concerned with the output, merely the execution time
    # Have to explicitly map to time function, other wise
/bin/bash/time will be used
    /usr/bin/time -f "Command:        %C\nUser Time:
%U(sec)\nSystem Time:    %S(sec)\nMaximum Memory: %M(KB)\nExit Status:
%x" $proverifdir/./analyzer -in pi
$specsdir/proverif/NSL_full_2_proverif_pi.txt > /dev/null
done

echo "End Test:   Execution Time: Proverif-NSL Full 2" >>
$resultsdir/proverif/NSL_full_2_proverif_pi_exec.txt
```

Listing 16: ProVerif Spi Calculus NSL Full 2 Protocol Test Script

Listing 17 is the execution time and main memory requirement test script for ProVerif for the Kerberos protocol.

```
#!/bin/bash
# File:        kerberos_proverif_pi.sh
# Created By:  Chris W. Hoffmeister
# Created On:  05 Mar 2007
# Modified By: -
# Modified On: -
# Description: Shell script to test Execution Time of
kerberos_proverif_pi.txt

# Setup variables
proverifdir=/usr/bin/proverif1.13pl7
```

```
resultsdir=/share/thesis/data/results
specsdir=/share/thesis/data/specifications
trials=5 # Default number of experiments to run

# Determines if the number of trials was passed in
if [ $# -eq 1 ]
then
# Number of trials was passed in
    trials=$1
fi

echo "Begin Test: Execution Time: Proverif-Kerberos" >
$resultsdir/proverif/kerberos_proverif_pi_exec.txt
date >> $resultsdir/proverif/kerberos_proverif_pi_exec.txt

# Perform trials
for ((trial=0; trial<trials; trial=trial+1)) do
    echo "Trial:  $trial" >>
$resultsdir/proverif/kerberos_proverif_pi_exec.txt
    # We're not concerned with the output, merely the execution time
    # Have to explicitly map to time function, other wise
/bin/bash/time will be used
    /usr/bin/time -f "Command:        %C\nUser Time:
%U(sec)\nSystem Time:    %S(sec)\nMaximum Memory: %M(KB)\nExit Status:
%x" $proverifdir/./analyzer -in pi
$specsdir/proverif/kerberos_proverif_pi.txt > /dev/null
done

echo "End Test:   Execution Time: Proverif-Kerberos" >>
$resultsdir/proverif/kerberos_proverif_pi_exec.txt
```

Listing 17: ProVerif Spi Calculus Kerberos Protocol Test Script

b. Secondary Memory Requirement

Listing 18 is the secondary memory requirement test script for ProVerif.

```
#!/bin/bash
# File:  proverif_second_mem.sh
# Created By: Chris W. Hoffmeister
```

```
# Created On: 06 Feb 2007
# Modified On: -
# Modified By: -
# Description: Script that runs ls to determine ProVerif size on disk

# Setup variables
tooldir=/usr/bin/proverif1.13pl7
resultsdir=/share/thesis/data/results

# Perform experiment
ls -lkR $tooldir | grep -F "$tooldir
total " > $resultsdir/proverif/proverif_second_mem.txt
```

Listing 18: ProVerif Secondary Memory Requirement Test Script

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C: EVALUATION DATA

Here we present the raw data that was collected during the evaluation of CPSA and ProVerif and the calculated data used in the evaluation methodology.

A. CPSA

1. Observed Correctness Criteria

a. Confidentiality & Authentication

Listing 19 is the collected data for CPSA for the simplified version of the NS protocol. The gray background portions of the listing show that CPSA identified that there exists a weakness in the protocol. Specifically the highlighted sections show that a man in the middle attack exists in the NS protocol.

```
CPSA> (*
>  * File:          NS_simple_2_cpsa.cpsa
>  * Created By:    Chris W. Hoffmeister
>  * Created On:    21 Jan 2007
>  * Modified By:   -
>  * Modified On:   -
>  * Description:   Simplified specification for the Needham-Schroeder
Protocol 2,
>  *               from [NEED1978] for the Cryptographic Protocol
Shapes Analyzer
>  *               (CPSA) protocol analysis system. This specification
is meant
>  *               to be run via the batch mode of CPSA.
>  *)
>
> (*
>  * Protocol Specification
>  *)
> protocol ns_simple_2=(
>
> (*
>  * Initiator Role, Alice
```

```
>  *)
>     role init(A:name; B:name; Na:nonce; Nb:nonce)=(
>         non=(privk(A))
>         uniq=(Na)
>         messages=(
>             + {A, Na}pubk(B);
>             - {Na, Nb}pubk(A);
>             + {Nb}pubk(B);
>         );
>     )
>
> (*
>  * Responder Role, Bob
>  *)
>     role resp(A:name; B:name; Na:nonce; Nb:nonce)=(
>         non=(privk(B))
>         uniq=(Nb)
>         messages=(
>             - {A, Na}pubk(B);
>             + {Na, Nb}pubk(A);
>             - {Nb}pubk(B);
>         )
>     )
> )
CPSA:ns_simple_2>
CPSA:ns_simple_2> (*
>  * Analyzer Commands
>  *)
> (*
>  * Test the initiator role, Alice
>  *)
> create init[3]
instantiated strand

CPSA:ns_simple_2*> non privk(B) (* Assume the single instance of the
responder is honest *)
added to non
```

```
CPSA:ns_simple_2*> printskel
unrealized input(A:name;Na:nonce;B:name;Nb:nonce) =
(
   strands = (
      original = init(A;B;Na;Nb)[3];
      );

   non = (privk(A);privk(B));
   uniq = (Na);
   safe = ();
   order = (

      );
   )

CPSA:ns_simple_2*> discover
realized output(B:name;A:name;Na:nonce;Nb:nonce) =
(
   strands = (
      original = init(A;B;Na;Nb)[3];
      resp-1 = resp(A;B;Na;Nb)[2];
      );

   non = (privk(A);privk(B));
   uniq = (Na;Nb);
   safe = (privk(A);privk(B);Na;Nb);
   order = (
      original[1] <= resp-1[1];
      resp-1[2] <= original[2];
      );
   )

CPSA:ns_simple_2*> (*
>  * Test the responder role, Bob
>  *)
> create resp[3]
```

```
instantiated strand

CPSA:ns_simple_2*> non privk(A) (* Assume the single instance of the
initiator is honest *)
added to non

CPSA:ns_simple_2*> printskel
unrealized input(A:name;Na:nonce;B:name;Nb:nonce) =
(
   strands = (
      original = resp(A;B;Na;Nb)[3];
      );

   non = (privk(B);privk(A));
   uniq = (Nb);
   safe = ();
   order = (

      );
   )

CPSA:ns_simple_2*> discover
realized output(B:name;A:name;Na:nonce;B1:name;Nb:nonce) =
(
   strands = (
      original = resp(A;B;Na;Nb)[3];
      init-1 = init(A;B1;Na;Nb)[3];
      );

   non = (privk(B);privk(A));
   uniq = (Nb;Na);
   safe = (privk(B);privk(A));
   order = (
      original[2] <= init-1[2];
      init-1[1] <= original[1];
      init-1[3] <= original[3];
      );
```

```
    )
```

Listing 19: CPSA NS Simple 2 Observed Correctness Test Data

Listing 20 is the data collected for CPSA for the full version of the NS protocol. Again the gray background portion of the listing shows that CPSA identified that there exists a weakness in the protocol. The highlighted portions also show that Alice and Bob need not use the exact same servers to get each others certificates.

```
CPSA> (*
>  * File:         NS_full_2_cpsa.cpsa
>  * Created By:   Chris W. Hoffmeister
>  * Created On:   21 Jan 2007
>  * Modified By:  -
>  * Modified On:  -
>  * Description:  Full specification for the Needham-Schroeder
Protocol 2, from
>  *               [NEED1978] for the Cryptographic Protocol Shapes
Analyzer
>  *               (CPSA) protocol analysis system. This specification
is meant
>  *               to be run via the batch mode of CPSA.
>  *)
>
> (*
>  * Protocol Specification
>  *)
> protocol ns_full_2=(
>
> (*
>  * Initiator Role, Alice
>  *)
>     role init(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
>         non=(privk(A))
>         uniq=(Na)
>         messages=(
>             + (A, B);
>             - {B, pubk(B)}privk(S);
>             + {A, Na}pubk(B);
```

```
>             - {Na, Nb}pubk(A);
>             + {Nb}pubk(B);
>         )
>     )
>
> (*
>  * Responder Role, Bob
>  *)
>     role resp(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
>         non=(privk(B))
>         uniq=(Nb)
>         messages=(
>             - {A, Na}pubk(B);
>             + (B, A);
>             - {A, pubk(A)}privk(S);
>             + {Na, Nb}pubk(A);
>             - {Nb}pubk(B);
>         )
>     )
>
> (*
>  * Server Role, Server
>  *)
>     role serv(A:name; B:name; S:name)=(
>         non=(privk(S))
>         uniq=()
>         messages=(
>             - (A, B);
>             + {B, pubk(B)}privk(S);
>             - (B, A);
>             + {A, pubk(A)}privk(S);
>         )
>     )
> )
CPSA:ns_full_2>
CPSA:ns_full_2> (*
>  * Analyzer Commands
```

```
>  *)
> (*
>  * Test the initiator role, Alice
>  *)
> create init[5]
instantiated strand

CPSA:ns_full_2*> non privk(B) (* Assume the single instance of the
responder is honest *)
added to non

CPSA:ns_full_2*> printskel
unrealized input(S:name;A:name;Na:nonce;B:name;Nb:nonce) =
(
   strands = (
      original = init(A;B;S;Na;Nb)[5];
      );

   non = (privk(A);privk(B));
   uniq = (Na);
   safe = ();
   order = (

      );
   )

CPSA:ns_full_2*> discover
realized output(S:name;B:name;S1:name;A:name;Na:nonce;Nb:nonce) =
(
   strands = (
      original = init(A;B;S;Na;Nb)[5];
      resp-1 = resp(A;B;S1;Na;Nb)[4];
      );

   non = (privk(A);privk(B));
   uniq = (Na;Nb);
   safe = (privk(A);privk(B);Na;Nb);
```

```
   order = (
      original[3] <= resp-1[1];
      resp-1[4] <= original[4];
      );
   )

CPSA:ns_full_2*> (*
>  * Test the responder role, Bob
>  *)
> create resp[5]
instantiated strand

CPSA:ns_full_2*> non privk(A) (* Assume the single instance of the
initiator is honest *)
added to non

CPSA:ns_full_2*> printskel
unrealized input(S:name;A:name;Na:nonce;B:name;Nb:nonce) =
(
   strands = (
      original = resp(A;B;S;Na;Nb)[5];
      );

   non = (privk(B);privk(A));
   uniq = (Nb);
   safe = ();
   order = (

      );
   )

CPSA:ns_full_2*> discover
realized output(S:name;B:name;S1:name;A:name;Na:nonce;B1:name;Nb:nonce)
=
(
   strands = (
      original = resp(A;B;S;Na;Nb)[5];
```

```
      init-1 = init(A;B1;S1;Na;Nb)[5];
      );

   non = (privk(B);privk(A));
   uniq = (Nb;Na);
   safe = (privk(B);privk(A));
   order = (
      original[4] <= init-1[4];
      init-1[3] <= original[1];
      init-1[5] <= original[5];
      );
   )
```

Listing 20: CPSA NS Full 2 Observed Correctness Test Data

Listing 21 lists the data from the CPSA NSL Simple 2 Observed Correctness Test. The gray background section shows that the only strands possible are from the expected participants and therefore that the protocol is secure.

```
CPSA> (*
>  * File:         NSL_simple_2_cpsa.cpsa
>  * Created By:   Chris W. Hoffmeister
>  * Created On:   21 Jan 2007
>  * Modified By:  -
>  * Modified On:  -
>  * Description:  Simplified specification for the Needham-Schroeder-
Lowe
>  *               Protocol from [LOWE1996] for the Cryptographic
Protocol
>  *               Shapes Analyzer (CPSA) protocol analysis system.
This
>  *               specification is meant to be run via the batch mode
of CPSA.
>  *)
>
> (*
>  * Protocol Specification
>  *)
> protocol nsl_simple_2=(
```

```
>
> (*
>  * Initiator Role, Alice
>  *)
>     role init(A:name; B:name; Na:nonce; Nb:nonce)=(
>         non=(privk(A))
>         uniq=(Na)
>         messages=(
>             + {A, Na}pubk(B);
>             - {B, Na, Nb}pubk(A);
>             + {Nb}pubk(B);
>         );
>     )
>
> (*
>  * Responder Role, Bob
>  *)
>     role resp(A:name; B:name; Na:nonce; Nb:nonce)=(
>         non=(privk(B))
>         uniq=(Nb)
>         messages=(
>             - {A, Na}pubk(B);
>             + {B, Na, Nb}pubk(A);
>             - {Nb}pubk(B);
>         )
>     )
> )
CPSA:nsl_simple_2>
CPSA:nsl_simple_2> (*
>  * Analyzer Commands
>  *)
> (*
>  * Test the initiator role, Alice
>  *)
> create init[3]
instantiated strand
```

```
CPSA:nsl_simple_2*> non privk(B) (* Assume the single instance of the
responder is honest *)
added to non

CPSA:nsl_simple_2*> printskel
unrealized input(A:name;Na:nonce;B:name;Nb:nonce) =
   (
   strands = (
      original = init(A;B;Na;Nb)[3];
      );

   non = (privk(A);privk(B));
   uniq = (Na);
   safe = ();
   order = (

      );
   )

CPSA:nsl_simple_2*> discover
realized output(A:name;B:name;Na:nonce;Nb:nonce) =
   (
   strands = (
      original = init(A;B;Na;Nb)[3];
      resp-1 = resp(A;B;Na;Nb)[2];
      );

   non = (privk(A);privk(B));
   uniq = (Na;Nb);
   safe = (privk(A);privk(B);Na;Nb);
   order = (
      original[1] <= resp-1[1];
      resp-1[2] <= original[2];
      );
   )

CPSA:nsl_simple_2*> (*
```

```
>  * Test the responder role, Bob
>  *)
> create resp[3]
instantiated strand

CPSA:nsl_simple_2*> non privk(A) (* Assume the single instance of the
initiator is honest *)
added to non

CPSA:nsl_simple_2*> printskel
unrealized input(A:name;Na:nonce;B:name;Nb:nonce) =
   (
   strands = (
      original = resp(A;B;Na;Nb)[3];
      );

   non = (privk(B);privk(A));
   uniq = (Nb);
   safe = ();
   order = (

      );
   )

CPSA:nsl_simple_2*> discover
realized output(A:name;Na:nonce;B:name;Nb:nonce) =
   (
   strands = (
      original = resp(A;B;Na;Nb)[3];
      init-1 = init(A;B;Na;Nb)[3];
      );

   non = (privk(B);privk(A));
   uniq = (Nb;Na);
   safe = (privk(B);privk(A);Nb;Na);
   order = (
      original[2] <= init-1[2];
```

```
      init-1[1] <= original[1];
      init-1[3] <= original[3];
      );
   )
```

Listing 21: CPSA NSL Simple 2 Observed Correctness Test Data

Listing 22 lists the data from the CPSA NSL Full 2 Observed Correctness Test. The gray background section shows that the only strands possible are from the expected participants and therefore that the protocol is secure. As expected the highlighted section also shows that Alice and Bob do not need to use the same certificate server.

```
CPSA> (*
>  * File:         NSL_full_2_cpsa.cpsa
>  * Created By:   Chris W. Hoffmeister
>  * Created On:   21 Jan 2007
>  * Modified By:  -
>  * Modified On:  -
>  * Description:  Full specification for the Needham-Schroeder-Lowe
Protocol
>  *               from [LOWE1996] for the Cryptographic Protocol
Shapes Analyzer
>  *               (CPSA) protocol analysis system. This specification
is meant
>  *               to be run via the batch mode of CPSA.
>  *)
>
> (*
>  * Protocol Specification
>  *)
> protocol nsl_full_2=(
>
> (*
>  * Initiator Role, Alice
>  *)
>     role init(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
>         non=(privk(A))
```

```
>         uniq=(Na)
>         messages=(
>             + (A, B);
>             - {B, pubk(B)}privk(S);
>             + {A, Na}pubk(B);
>             - {B, Na, Nb}pubk(A);
>             + {Nb}pubk(B);
>         )
>     )
>
> (*
>  * Responder Role, Bob
>  *)
>     role resp(A:name; B:name; S:name; Na:nonce; Nb:nonce)=(
>         non=(privk(B))
>         uniq=(Nb)
>         messages=(
>             - {A, Na}pubk(B);
>             + (B, A);
>             - {A, pubk(A)}privk(S);
>             + {B, Na, Nb}pubk(A);
>             - {Nb}pubk(B);
>         )
>     )
>
> (*
>  * Server Role, Server
>  *)
>     role serv(A:name; B:name; S:name)=(
>         non=(privk(S))
>         uniq=()
>         messages=(
>             - (A, B);
>             + {B, pubk(B)}privk(S);
>             - (B, A);
>             + {A, pubk(A)}privk(S);
>         )
```

```
>     )
> )
CPSA:nsl_full_2>
CPSA:nsl_full_2> (*
>  * Analyzer Commands
>  *)
> (*
>  * Test the initiator role, Alice
>  *)
> create init[5]
instantiated strand

CPSA:nsl_full_2*> non privk(B) (* Assume the single instance of the
responder is honest *)
added to non

CPSA:nsl_full_2*> printskel
unrealized input(S:name;A:name;Na:nonce;B:name;Nb:nonce) =
   (
   strands = (
      original = init(A;B;S;Na;Nb)[5];
      );

   non = (privk(A);privk(B));
   uniq = (Na);
   safe = ();
   order = (

      );
   )

CPSA:nsl_full_2*> discover
realized output(S:name;S1:name;A:name;B:name;Na:nonce;Nb:nonce) =
   (
   strands = (
      original = init(A;B;S;Na;Nb)[5];
      resp-1 = resp(A;B;S1;Na;Nb)[4];
```

```
      );

   non = (privk(A);privk(B));
   uniq = (Na;Nb);
   safe = (privk(A);privk(B);Na;Nb);
   order = (
      original[3] <= resp-1[1];
      resp-1[4] <= original[4];
      );
   )

CPSA:nsl_full_2*> (*
>  * Test the responder role, Bob
>  *)
> create resp[5]
instantiated strand

CPSA:nsl_full_2*> non privk(A) (* Assume the single instance of the
initiator is honest *)
added to non

CPSA:nsl_full_2*> printskel
unrealized input(S:name;A:name;Na:nonce;B:name;Nb:nonce) =
   (
   strands = (
      original = resp(A;B;S;Na;Nb)[5];
      );

   non = (privk(B);privk(A));
   uniq = (Nb);
   safe = ();
   order = (

      );
   )

CPSA:nsl_full_2*> discover
```

```
realized output(S:name;S1:name;A:name;Na:nonce;B:name;Nb:nonce) =
   (
   strands = (
      original = resp(A;B;S;Na;Nb)[5];
      init-1 = init(A;B;S1;Na;Nb)[5];
      );

   non = (privk(B);privk(A));
   uniq = (Nb;Na);
   safe = (privk(B);privk(A);Nb;Na);
   order = (
      original[4] <= init-1[4];
      init-1[3] <= original[1];
      init-1[5] <= original[5];
      );
   )
```

Listing 22: CPSA NSL Full 2 Observed Correctness Test Data

2. Performance Criteria

a. Execution Time

Table 31 presents the execution time data collected for the full version of the NS Protocol 2 as discussed in [NEED1978] for CPSA. From this the CI^{+} value for the execution time was calculated; the Execution Time CI^{+} value is 5.49 sec.

Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)
0	5.48	10	5.49	20	5.49	30	5.49	40	5.6
1	5.48	11	5.49	21	5.5	31	5.49	41	5.48
2	5.47	12	5.49	22	5.48	32	5.48	42	5.48
3	5.49	13	5.5	23	5.49	33	5.49	43	5.48
4	5.47	14	5.48	24	5.49	34	5.49	44	5.48
5	5.48	15	5.49	25	5.47	35	5.49	45	5.5
6	5.48	16	5.5	26	5.47	36	5.48	46	5.49
7	5.47	17	5.5	27	5.51	37	5.48	47	5.49
8	5.48	18	5.48	28	5.48	38	5.48	48	5.48
9	5.48	19	5.49	29	5.48	39	5.49	49	5.47
-	-	-	-	-	-	-	-	50	5.48

Table 31: CPSA NS Full 2 Execution Criterion Test Data

Table 32 presents the execution time data collected for the full version of the NSL Protocol 2 as discussed in [LOWE1996] for CPSA. From this the CI^+ value for the execution time was calculated; the Execution Time CI^+ value is 13.94 sec.

Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)
0	5.48	10	5.49	20	5.49	30	5.49	40	5.6
1	5.48	11	5.49	21	5.5	31	5.49	41	5.48
2	5.47	12	5.49	22	5.48	32	5.48	42	5.48
3	5.49	13	5.5	23	5.49	33	5.49	43	5.48
4	5.47	14	5.48	24	5.49	34	5.49	44	5.48
5	5.48	15	5.49	25	5.47	35	5.49	45	5.5
6	5.48	16	5.5	26	5.47	36	5.48	46	5.49
7	5.47	17	5.5	27	5.51	37	5.48	47	5.49
8	5.48	18	5.48	28	5.48	38	5.48	48	5.48
9	5.48	19	5.49	29	5.48	39	5.49	49	5.47
-	-	-	-	-	-	-	-	50	5.48

Table 32: CPSA NSL Full 2 Execution Criterion Test Data

b. Secondary Memory Requirement

Listing 23 presents the secondary memory requirement data collected for CPSA. The total main memory required for CPSA is 8.348 MB.

```
/usr/local/cpsa-0.81:
total 456
/usr/local/cpsa-0.81/doc:
total 348
/usr/local/cpsa-0.81/protocols:
total 112
/usr/local/cpsa-0.81/protocols/batch-scripts:
total 40
/usr/local/cpsa-0.81/src:
total 48
/usr/local/cpsa-0.81/src/cpsa:
total 5320
/usr/local/cpsa-0.81/src/cpsa/ounit-1.0.2:
total 12
```

```
/usr/local/cpsa-0.81/src/extlib-1.4:
total 1596
/usr/local/cpsa-0.81/src/make:
total 100
/usr/local/cpsa-0.81/src/ounit-1.0.2:
total 240
/usr/local/cpsa-0.81/src/ounit-1.0.2/doc:
total 16
/usr/local/cpsa-0.81/src/ounit-1.0.2/doc/manual:
total 8
/usr/local/cpsa-0.81/src/ounit-1.0.2/doc/manual/src:
total 12
/usr/local/cpsa-0.81/src/ounit-1.0.2/examples:
total 40
```

Listing 23: CPSA Secondary Memory Requirement Criterion Test Data

c. Main Memory Requirement

Table 33 presents the main memory requirement data collected for the full version of the NS Protocol 2 as discussed in [NEED1978] for CPSA. From this the CI^+ value for the main memory requirement was calculated; the Main Memory CI^+ value is 0 KB.

Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)
0	0	10	0	20	0	30	0	40	0
1	0	11	0	21	0	31	0	41	0
2	0	12	0	22	0	32	0	42	0
3	0	13	0	23	0	33	0	43	0
4	0	14	0	24	0	34	0	44	0
5	0	15	0	25	0	35	0	45	0
6	0	16	0	26	0	36	0	46	0
7	0	17	0	27	0	37	0	47	0
8	0	18	0	28	0	38	0	48	0
9	0	19	0	29	0	39	0	49	0
-	-	-	-	-	-	-	-	50	0

Table 33: CPSA NS Full 2 Main Memory Requirement Criterion Test Data

Table 34 presents the main memory requirement data collected for the full version of the NSL Protocol 2 as discussed in [LOWE1996] for CPSA. From this the CI^+ value for the main memory requirement was calculated; the Main Memory CI^+ value is 0 KB.

Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)
0	0	10	0	20	0	30	0	40	0
1	0	11	0	21	0	31	0	41	0
2	0	12	0	22	0	32	0	42	0
3	0	13	0	23	0	33	0	43	0
4	0	14	0	24	0	34	0	44	0
5	0	15	0	25	0	35	0	45	0
6	0	16	0	26	0	36	0	46	0
7	0	17	0	27	0	37	0	47	0
8	0	18	0	28	0	38	0	48	0
9	0	19	0	29	0	39	0	49	0
-	-	-	-	-	-	-	-	50	0

Table 34: CPSA NSL Full 2 Main Memory Requirement Criterion Test Data

B. PROVERIF

1. Observed Correctness Criteria

a. Confidentiality & Authentication

Listing 24 is the collected data for Proverif for the full version of the NS protocol. The gray background portions of the listing show that Proverif identified that there exists a weakness in the protocol.

```
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_40,v_40
Rule 1: attacker:sign(x_42,y_43) -> attacker:x_42
Rule 2: attacker:sencrypt(x_44,y_45) & attacker:y_45 -> attacker:x_44
Rule 3: attacker:sign(x_46,y_47) & attacker:pk(y_47) -> attacker:x_46
Rule 4: attacker:v_49 & attacker:v_48 -> attacker:encrypt(v_49,v_48)
Rule 5: attacker:v_51 & attacker:v_50 -> attacker:sign(v_51,v_50)
Rule 6: attacker:v_52 -> attacker:host(v_52)
```

```
Rule 7: attacker:v_53 -> attacker:pk(v_53)
Rule 8: attacker:encrypt(x_54,pk(y_55)) & attacker:y_55 ->
attacker:x_54
Rule 9: attacker:v_57 & attacker:v_56 -> attacker:sencrypt(v_57,v_56)
Rule 10: attacker:v_59 & attacker:v_58 -> attacker:(v_59,v_58)
Rule 11: attacker:(v_61,v_60) -> attacker:v_61
Rule 12: attacker:(v_63,v_62) -> attacker:v_62
Rule 13: mess:v_65,v_64 & attacker:v_65 -> attacker:v_64
Rule 14: attacker:v_67 & attacker:v_66 -> mess:v_67,v_66
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_68]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: attacker:v_71 -> attacker:(host(pk(skA[])),v_71)
Rule 23: attacker:sign((v_78,v_79),skS[]) & attacker:v_79 ->
attacker:encrypt((Na[sign((v_78,v_79),skS[]),v_79,sid_80],host(pk(skA[]
))),v_78)
Rule 24:
attacker:encrypt((Na[sign((v_86,v_87),skS[]),v_87,sid_88],v_89),pk(skA[
])) & attacker:sign((v_86,v_87),skS[]) & attacker:v_87 ->
attacker:encrypt(v_89,v_86)
Rule 25:
attacker:encrypt((Na[sign((v_90,host(pk(skB[]))),skS[]),host(pk(skB[]))
,sid_91],v_92),pk(skA[])) & attacker:sign((v_90,host(pk(skB[]))),skS[])
& attacker:host(pk(skB[])) ->
end:sid_91,endAfull(Na[sign((v_90,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_91],host(pk(skA[])),host(pk(skB[])),v_90,pk(skA[]),v_92)
Rule 26:
attacker:encrypt((Na[sign((v_93,host(pk(skB[]))),skS[]),host(pk(skB[]))
,sid_94],v_95),pk(skA[])) & attacker:sign((v_93,host(pk(skB[]))),skS[])
& attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_93,host(pk(skB[]))),skS[]),hos
t(pk(skB[])),sid_94])
```

```
Rule 27:
attacker:encrypt((Na[sign((v_96,host(pk(skB[]))),skS[]),host(pk(skB[]))
,sid_97],v_98),pk(skA[])) & attacker:sign((v_96,host(pk(skB[]))),skS[])
& attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_98)
Rule 28: attacker:encrypt((v_105,v_106),pk(skB[])) ->
attacker:(host(pk(skB[])),v_106)
Rule 29:
begin:beginAfull(v_115,v_114,host(pk(skB[])),pk(skB[]),v_113,Nb[sign((v
_113,v_114),skS[]),encrypt((v_115,v_114),pk(skB[])),sid_116]), ms_26 =
sign((v_113,v_114),skS[]), m_23 = encrypt((v_115,v_114),pk(skB[])),
sid_99 = sid_116 & attacker:sign((v_113,v_114),skS[]) &
attacker:encrypt((v_115,v_114),pk(skB[])) ->
attacker:encrypt((v_115,Nb[sign((v_113,v_114),skS[]),encrypt((v_115,v_1
14),pk(skB[])),sid_116]),v_113)
Rule 30:
attacker:encrypt(Nb[sign((v_120,host(pk(skA[]))),skS[]),encrypt((v_121,
host(pk(skA[]))),pk(skB[])),sid_122],pk(skB[])) &
begin:beginAfull(v_121,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_120,
Nb[sign((v_120,host(pk(skA[]))),skS[]),encrypt((v_121,host(pk(skA[]))),
pk(skB[])),sid_122]), m3_28 =
encrypt(Nb[sign((v_120,host(pk(skA[]))),skS[]),encrypt((v_121,host(pk(s
kA[]))),pk(skB[])),sid_122],pk(skB[])), ms_26 =
sign((v_120,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_121,host(pk(skA[]))),pk(skB[])), sid_99 = sid_122 &
attacker:sign((v_120,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_121,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_121)
Rule 31:
attacker:encrypt(Nb[sign((v_123,host(pk(skA[]))),skS[]),encrypt((v_124,
host(pk(skA[]))),pk(skB[])),sid_125],pk(skB[])) &
begin:beginAfull(v_124,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_123,
Nb[sign((v_123,host(pk(skA[]))),skS[]),encrypt((v_124,host(pk(skA[]))),
pk(skB[])),sid_125]), m3_28 =
encrypt(Nb[sign((v_123,host(pk(skA[]))),skS[]),encrypt((v_124,host(pk(s
kA[]))),pk(skB[])),sid_125],pk(skB[])), ms_26 =
sign((v_123,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_124,host(pk(skA[]))),pk(skB[])), sid_99 = sid_125 &
```

```
attacker:sign((v_123,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_124,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_123,host(pk(skA[]))),skS[]),en
crypt((v_124,host(pk(skA[]))),pk(skB[])),sid_125])
Rule 32: attacker:(v_131,host(x_132)) ->
attacker:sign((x_132,host(x_132)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query evinj:endAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39) ==>
evinj:beginAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39)
goal reachable:
begin:beginAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB
[])),endsid_620],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb
[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),ho
st(pk(skB[]))),skS[]),host(pk(skB[])),endsid_620],host(pk(skA[]))),pk(s
kB[])),sid_621]), ms_26 = sign((pk(skA[]),host(pk(skA[]))),skS[]), m_23
=
encrypt((Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),end
sid_620],host(pk(skA[]))),pk(skB[])), sid_99 = sid_621 ->
end:endsid_620,endAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host
(pk(skB[])),endsid_620],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(sk
A[]),Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(sk
B[]),host(pk(skB[]))),skS[]),host(pk(skB[])),endsid_620],host(pk(skA[])
)),pk(skB[])),sid_621])
RESULT evinj:endAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39) ==>
evinj:beginAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_639,v_639
Rule 1: attacker:sign(x_641,y_642) -> attacker:x_641
Rule 2: attacker:sencrypt(x_643,y_644) & attacker:y_644 ->
attacker:x_643
Rule 3: attacker:sign(x_645,y_646) & attacker:pk(y_646) ->
attacker:x_645
```

```
Rule 4: attacker:v_648 & attacker:v_647 ->
attacker:encrypt(v_648,v_647)
Rule 5: attacker:v_650 & attacker:v_649 -> attacker:sign(v_650,v_649)
Rule 6: attacker:v_651 -> attacker:host(v_651)
Rule 7: attacker:v_652 -> attacker:pk(v_652)
Rule 8: attacker:encrypt(x_653,pk(y_654)) & attacker:y_654 ->
attacker:x_653
Rule 9: attacker:v_656 & attacker:v_655 ->
attacker:sencrypt(v_656,v_655)
Rule 10: attacker:v_658 & attacker:v_657 -> attacker:(v_658,v_657)
Rule 11: attacker:(v_660,v_659) -> attacker:v_660
Rule 12: attacker:(v_662,v_661) -> attacker:v_661
Rule 13: mess:v_664,v_663 & attacker:v_664 -> attacker:v_663
Rule 14: attacker:v_666 & attacker:v_665 -> mess:v_666,v_665
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_667]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: attacker:v_670 -> attacker:(host(pk(skA[])),v_670)
Rule 23: attacker:sign((v_677,v_678),skS[]) & attacker:v_678 ->
attacker:encrypt((Na[sign((v_677,v_678),skS[]),v_678,sid_679],host(pk(s
kA[]))),v_677)
Rule 24:
attacker:encrypt((Na[sign((v_685,v_686),skS[]),v_686,sid_687],v_688),pk
(skA[])) & attacker:sign((v_685,v_686),skS[]) & attacker:v_686 ->
attacker:encrypt(v_688,v_685)
Rule 25:
attacker:encrypt((Na[sign((v_689,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_690],v_691),pk(skA[])) &
attacker:sign((v_689,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
-> end:sid_690,endAparam(host(pk(skA[])))
Rule 26:
attacker:encrypt((Na[sign((v_692,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_693],v_694),pk(skA[])) &
```

```
attacker:sign((v_692,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
->
attacker:sencrypt(secretANa[],Na[sign((v_692,host(pk(skB[]))),skS[]),ho
st(pk(skB[])),sid_693])
Rule 27:
attacker:encrypt((Na[sign((v_695,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_696],v_697),pk(skA[])) &
attacker:sign((v_695,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
-> attacker:sencrypt(secretANb[],v_697)
Rule 28: begin:beginAparam(v_705), m_23 =
encrypt((v_704,v_705),pk(skB[])), sid_99 = sid_706 &
attacker:encrypt((v_704,v_705),pk(skB[])) ->
attacker:(host(pk(skB[])),v_705)
Rule 29: attacker:sign((v_712,v_713),skS[]) & begin:beginAparam(v_713),
ms_26 = sign((v_712,v_713),skS[]), m_23 =
encrypt((v_714,v_713),pk(skB[])), sid_99 = sid_715 &
attacker:encrypt((v_714,v_713),pk(skB[])) ->
attacker:encrypt((v_714,Nb[sign((v_712,v_713),skS[]),encrypt((v_714,v_7
13),pk(skB[])),sid_715]),v_712)
Rule 30:
attacker:encrypt(Nb[sign((v_719,host(pk(skA[]))),skS[]),encrypt((v_720,
host(pk(skA[]))),pk(skB[])),sid_721],pk(skB[])) &
attacker:sign((v_719,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))), m3_28 =
encrypt(Nb[sign((v_719,host(pk(skA[]))),skS[]),encrypt((v_720,host(pk(s
kA[]))),pk(skB[])),sid_721],pk(skB[])), ms_26 =
sign((v_719,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_720,host(pk(skA[]))),pk(skB[])), sid_99 = sid_721 &
attacker:encrypt((v_720,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_720)
Rule 31:
attacker:encrypt(Nb[sign((v_722,host(pk(skA[]))),skS[]),encrypt((v_723,
host(pk(skA[]))),pk(skB[])),sid_724],pk(skB[])) &
attacker:sign((v_722,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))), m3_28 =
encrypt(Nb[sign((v_722,host(pk(skA[]))),skS[]),encrypt((v_723,host(pk(s
kA[]))),pk(skB[])),sid_724],pk(skB[])), ms_26 =
```

```
sign((v_722,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_723,host(pk(skA[]))),pk(skB[])), sid_99 = sid_724 &
attacker:encrypt((v_723,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_722,host(pk(skA[]))),skS[]),en
crypt((v_723,host(pk(skA[]))),pk(skB[])),sid_724])
Rule 32: attacker:(v_730,host(x_731)) ->
attacker:sign((x_731,host(x_731)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query evinj:endAparam(x_638) ==> evinj:beginAparam(x_638)
goal reachable: begin:beginAparam(host(pk(skA[]))), ms_26 =
sign((pk(skA[]),host(pk(skA[]))),skS[]), m_23 =
encrypt((Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),end
sid_1219],host(pk(skA[]))),pk(skB[])), sid_99 = sid_1220 ->
end:endsid_1219,endAparam(host(pk(skA[])))
RESULT evinj:endAparam(x_638) ==> evinj:beginAparam(x_638) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_1233,v_1233
Rule 1: attacker:sign(x_1235,y_1236) -> attacker:x_1235
Rule 2: attacker:sencrypt(x_1237,y_1238) & attacker:y_1238 ->
attacker:x_1237
Rule 3: attacker:sign(x_1239,y_1240) & attacker:pk(y_1240) ->
attacker:x_1239
Rule 4: attacker:v_1242 & attacker:v_1241 ->
attacker:encrypt(v_1242,v_1241)
Rule 5: attacker:v_1244 & attacker:v_1243 ->
attacker:sign(v_1244,v_1243)
Rule 6: attacker:v_1245 -> attacker:host(v_1245)
Rule 7: attacker:v_1246 -> attacker:pk(v_1246)
Rule 8: attacker:encrypt(x_1247,pk(y_1248)) & attacker:y_1248 ->
attacker:x_1247
Rule 9: attacker:v_1250 & attacker:v_1249 ->
attacker:sencrypt(v_1250,v_1249)
Rule 10: attacker:v_1252 & attacker:v_1251 -> attacker:(v_1252,v_1251)
```

```
Rule 11: attacker:(v_1254,v_1253) -> attacker:v_1254
Rule 12: attacker:(v_1256,v_1255) -> attacker:v_1255
Rule 13: mess:v_1258,v_1257 & attacker:v_1258 -> attacker:v_1257
Rule 14: attacker:v_1260 & attacker:v_1259 -> mess:v_1260,v_1259
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_1261]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: attacker:v_1264 -> attacker:(host(pk(skA[])),v_1264)
Rule 23: attacker:sign((v_1271,v_1272),skS[]) & attacker:v_1272 ->
attacker:encrypt((Na[sign((v_1271,v_1272),skS[]),v_1272,sid_1273],host(
pk(skA[]))),v_1271)
Rule 24:
begin:beginBfull(Na[sign((v_1279,v_1280),skS[]),v_1280,sid_1281],host(p
k(skA[])),v_1280,v_1279,pk(skA[]),v_1282), m_32 =
encrypt((Na[sign((v_1279,v_1280),skS[]),v_1280,sid_1281],v_1282),pk(skA
[])), ms_30 = sign((v_1279,v_1280),skS[]), hostX_29 = v_1280, sid_69 =
sid_1281 &
attacker:encrypt((Na[sign((v_1279,v_1280),skS[]),v_1280,sid_1281],v_128
2),pk(skA[])) & attacker:sign((v_1279,v_1280),skS[]) & attacker:v_1280
-> attacker:encrypt(v_1282,v_1279)
Rule 25:
begin:beginBfull(Na[sign((v_1283,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_1284],host(pk(skA[])),host(pk(skB[])),v_1283,pk(skA[]),v_1285),
m_32 =
encrypt((Na[sign((v_1283,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_12
84],v_1285),pk(skA[])), ms_30 = sign((v_1283,host(pk(skB[]))),skS[]),
hostX_29 = host(pk(skB[])), sid_69 = sid_1284 &
attacker:encrypt((Na[sign((v_1283,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1284],v_1285),pk(skA[])) &
attacker:sign((v_1283,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_1283,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_1284])
```

```
Rule 26:
begin:beginBfull(Na[sign((v_1286,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_1287],host(pk(skA[])),host(pk(skB[])),v_1286,pk(skA[]),v_1288),
m_32 =
encrypt((Na[sign((v_1286,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_12
87],v_1288),pk(skA[])), ms_30 = sign((v_1286,host(pk(skB[]))),skS[]),
hostX_29 = host(pk(skB[])), sid_69 = sid_1287 &
attacker:encrypt((Na[sign((v_1286,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1287],v_1288),pk(skA[])) &
attacker:sign((v_1286,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_1288)
Rule 27: attacker:encrypt((v_1295,v_1296),pk(skB[])) ->
attacker:(host(pk(skB[])),v_1296)
Rule 28: attacker:sign((v_1303,v_1304),skS[]) &
attacker:encrypt((v_1305,v_1304),pk(skB[])) ->
attacker:encrypt((v_1305,Nb[sign((v_1303,v_1304),skS[]),encrypt((v_1305
,v_1304),pk(skB[])),sid_1306]),v_1303)
Rule 29:
attacker:encrypt(Nb[sign((v_1310,host(pk(skA[]))),skS[]),encrypt((v_131
1,host(pk(skA[]))),pk(skB[])),sid_1312],pk(skB[])) &
attacker:sign((v_1310,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1311,host(pk(skA[]))),pk(skB[])) ->
end:sid_1312,endBfull(v_1311,host(pk(skA[])),host(pk(skB[])),pk(skB[]),
pk(skA[]),Nb[sign((v_1310,host(pk(skA[]))),skS[]),encrypt((v_1311,host(
pk(skA[]))),pk(skB[])),sid_1312])
Rule 30:
attacker:encrypt(Nb[sign((v_1313,host(pk(skA[]))),skS[]),encrypt((v_131
4,host(pk(skA[]))),pk(skB[])),sid_1315],pk(skB[])) &
attacker:sign((v_1313,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1314,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_1314)
Rule 31:
attacker:encrypt(Nb[sign((v_1316,host(pk(skA[]))),skS[]),encrypt((v_131
7,host(pk(skA[]))),pk(skB[])),sid_1318],pk(skB[])) &
attacker:sign((v_1316,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1317,host(pk(skA[]))),pk(skB[])) ->
```

```
attacker:sencrypt(secretBNb[],Nb[sign((v_1316,host(pk(skA[]))),skS[]),e
ncrypt((v_1317,host(pk(skA[]))),pk(skB[])),sid_1318])
Rule 32: attacker:(v_1324,host(x_1325)) ->
attacker:sign((x_1325,host(x_1325)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query
evinj:endBfull(x1_1227,x2_1228,x3_1229,x4_1230,x5_1231,x6_1232) ==>
evinj:beginBfull(x1_1227,x2_1228,x3_1229,x4_1230,x5_1231,x6_1232)
goal reachable:
begin:beginBfull(Na[sign((pk(y_1850),host(pk(y_1850))),skS[]),host(pk(y
_1850)),sid_1851],host(pk(skA[])),host(pk(y_1850)),pk(y_1850),pk(skA[])
,Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(y_1850
),host(pk(y_1850))),skS[]),host(pk(y_1850)),sid_1851],host(pk(skA[]))),
pk(skB[])),endsid_1852]), m_32 =
encrypt((Na[sign((pk(y_1850),host(pk(y_1850))),skS[]),host(pk(y_1850)),
sid_1851],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((
pk(y_1850),host(pk(y_1850))),skS[]),host(pk(y_1850)),sid_1851],host(pk(
skA[]))),pk(skB[])),endsid_1852]),pk(skA[])), ms_30 =
sign((pk(y_1850),host(pk(y_1850))),skS[]), hostX_29 = host(pk(y_1850)),
sid_69 = sid_1851 & attacker:y_1850 ->
end:endsid_1852,endBfull(Na[sign((pk(y_1850),host(pk(y_1850))),skS[]),h
ost(pk(y_1850)),sid_1851],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(
skA[]),Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(
y_1850),host(pk(y_1850))),skS[]),host(pk(y_1850)),sid_1851],host(pk(skA
[]))),pk(skB[])),endsid_1852])
rule 29
end:endsid_1941,endBfull(Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),h
ost(pk(y_1935)),sid_1932],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(
skA[]),Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(
y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),sid_1932],host(pk(skA
[]))),pk(skB[])),endsid_1941])
  rule 4
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
```

```
[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),sid_1932],h
ost(pk(skA[]))),pk(skB[])),endsid_1941],pk(skB[]))
    rule 8
attacker:Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((p
k(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),sid_1932],host(pk(s
kA[]))),pk(skB[])),endsid_1941]
      rule 24
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),sid_1932],h
ost(pk(skA[]))),pk(skB[])),endsid_1941],pk(y_1935))
        hypothesis
begin:beginBfull(Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y
_1935)),sid_1932],host(pk(skA[])),host(pk(y_1935)),pk(y_1935),pk(skA[])
,Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(y_1935
),host(pk(y_1935))),skS[]),host(pk(y_1935)),sid_1932],host(pk(skA[]))),
pk(skB[])),endsid_1941]), m_32 =
encrypt((Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),
sid_1932],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((
pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),sid_1932],host(pk(
skA[]))),pk(skB[])),endsid_1941]),pk(skA[])), ms_30 =
sign((pk(y_1935),host(pk(y_1935))),skS[]), hostX_29 = host(pk(y_1935)),
sid_69 = sid_1932
        rule 28
attacker:encrypt((Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(
y_1935)),sid_1932],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((
Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),sid_1932]
,host(pk(skA[]))),pk(skB[])),endsid_1941]),pk(skA[]))
          duplicate attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
          duplicate
attacker:encrypt((Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(
y_1935)),sid_1932],host(pk(skA[]))),pk(skB[]))
        duplicate attacker:sign((pk(y_1935),host(pk(y_1935))),skS[])
        duplicate attacker:host(pk(y_1935))
      hypothesis attacker:y_1935
    duplicate attacker:pk(skB[])
  rule 32 attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
    2-tuple attacker:(v_1870,host(pk(skA[])))
```

```
      any attacker:v_1870
      duplicate attacker:host(pk(skA[]))
  rule 4
attacker:encrypt((Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(
y_1935)),sid_1932],host(pk(skA[]))),pk(skB[]))
    2-tuple
attacker:(Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935))
,sid_1932],host(pk(skA[])))
      0-th
attacker:Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935)),
sid_1932]
        rule 8
attacker:(Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(y_1935))
,sid_1932],host(pk(skA[])))
          rule 23
attacker:encrypt((Na[sign((pk(y_1935),host(pk(y_1935))),skS[]),host(pk(
y_1935)),sid_1932],host(pk(skA[]))),pk(y_1935))
            rule 32 attacker:sign((pk(y_1935),host(pk(y_1935))),skS[])
              2-tuple attacker:(v_1878,host(pk(y_1935)))
                any attacker:v_1878
                duplicate attacker:host(pk(y_1935))
            rule 6 attacker:host(pk(y_1935))
              rule 7 attacker:pk(y_1935)
                hypothesis attacker:y_1935
          hypothesis attacker:y_1935
      rule 6 attacker:host(pk(skA[]))
        rule 17 attacker:pk(skA[])
    rule 18 attacker:pk(skB[])

A more detailed output of the traces is available with
  param traceDisplay = long.
Goal of the attack :
end:a_1[],endBfull(Na_9[],host(pk(skA_10[])),host(pk(skB_11[])),pk(skB_
11[]),pk(skA_10[]),Nb_12[])

out(c, pk(skA_10))
```

```
out(c, pk(skB_11))

out(c, pk(skS_8))

out(c, host(pk(skA_10)))

out(c, host(pk(skB_11)))

in(c, host(pk(a_2)))

event(beginBparam(host(pk(a_2))))

out(c, (host(pk(skA_10)),host(pk(a_2))))

in(c, (a_4,host(pk(skA_10))))

out(c, sign((pk(skA_10),host(pk(skA_10))),skS_8))

in(c, (a_6,host(pk(a_2))))

out(c, sign((pk(a_2),host(pk(a_2))),skS_8))

in(c, sign((pk(a_2),host(pk(a_2))),skS_8))

out(c, encrypt((Na_9,host(pk(skA_10))),pk(a_2)))

in(c, encrypt((Na_9,host(pk(skA_10))),pk(skB_11)))

event(beginAparam(host(pk(skA_10))))

out(c, (host(pk(skB_11)),host(pk(skA_10))))

in(c, sign((pk(skA_10),host(pk(skA_10))),skS_8))

event(beginAfull(Na_9,host(pk(skA_10)),host(pk(skB_11)),pk(skB_11),pk(s
kA_10),Nb_12))
```

```
out(c, encrypt((Na_9,Nb_12),pk(skA_10)))

in(c, encrypt((Na_9,Nb_12),pk(skA_10)))

event(beginBfull(Na_9,host(pk(skA_10)),host(pk(a_2)),pk(a_2),pk(skA_10)
,Nb_12))

out(c, encrypt(Nb_12,pk(a_2)))

in(c, encrypt(Nb_12,pk(skB_11)))

event(endBparam(host(pk(skB_11))))

event(endBfull(Na_9,host(pk(skA_10)),host(pk(skB_11)),pk(skB_11),pk(skA
_10),Nb_12))

An attack has been found.
RESULT evinj:endBfull(x1_1227,x2_1228,x3_1229,x4_1230,x5_1231,x6_1232)
==> evinj:beginBfull(x1_1227,x2_1228,x3_1229,x4_1230,x5_1231,x6_1232)
is false.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_2117,v_2117
Rule 1: attacker:sign(x_2119,y_2120) -> attacker:x_2119
Rule 2: attacker:sencrypt(x_2121,y_2122) & attacker:y_2122 ->
attacker:x_2121
Rule 3: attacker:sign(x_2123,y_2124) & attacker:pk(y_2124) ->
attacker:x_2123
Rule 4: attacker:v_2126 & attacker:v_2125 ->
attacker:encrypt(v_2126,v_2125)
Rule 5: attacker:v_2128 & attacker:v_2127 ->
attacker:sign(v_2128,v_2127)
Rule 6: attacker:v_2129 -> attacker:host(v_2129)
Rule 7: attacker:v_2130 -> attacker:pk(v_2130)
Rule 8: attacker:encrypt(x_2131,pk(y_2132)) & attacker:y_2132 ->
attacker:x_2131
```

```
Rule 9: attacker:v_2134 & attacker:v_2133 ->
attacker:sencrypt(v_2134,v_2133)
Rule 10: attacker:v_2136 & attacker:v_2135 -> attacker:(v_2136,v_2135)
Rule 11: attacker:(v_2138,v_2137) -> attacker:v_2138
Rule 12: attacker:(v_2140,v_2139) -> attacker:v_2139
Rule 13: mess:v_2142,v_2141 & attacker:v_2142 -> attacker:v_2141
Rule 14: attacker:v_2144 & attacker:v_2143 -> mess:v_2144,v_2143
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_2145]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: begin:beginBparam(v_2148), hostX_29 = v_2148, sid_69 =
sid_2149 & attacker:v_2148 -> attacker:(host(pk(skA[])),v_2148)
Rule 23: attacker:sign((v_2155,v_2156),skS[]) &
begin:beginBparam(v_2156), ms_30 = sign((v_2155,v_2156),skS[]),
hostX_29 = v_2156, sid_69 = sid_2157 & attacker:v_2156 ->
attacker:encrypt((Na[sign((v_2155,v_2156),skS[]),v_2156,sid_2157],host(
pk(skA[]))),v_2155)
Rule 24:
attacker:encrypt((Na[sign((v_2163,v_2164),skS[]),v_2164,sid_2165],v_216
6),pk(skA[])) & attacker:sign((v_2163,v_2164),skS[]) &
begin:beginBparam(v_2164), m_32 =
encrypt((Na[sign((v_2163,v_2164),skS[]),v_2164,sid_2165],v_2166),pk(skA
[])), ms_30 = sign((v_2163,v_2164),skS[]), hostX_29 = v_2164, sid_69 =
sid_2165 & attacker:v_2164 -> attacker:encrypt(v_2166,v_2163)
Rule 25:
attacker:encrypt((Na[sign((v_2167,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_2168],v_2169),pk(skA[])) &
attacker:sign((v_2167,host(pk(skB[]))),skS[]) &
begin:beginBparam(host(pk(skB[]))), m_32 =
encrypt((Na[sign((v_2167,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_21
68],v_2169),pk(skA[])), ms_30 = sign((v_2167,host(pk(skB[]))),skS[]),
hostX_29 = host(pk(skB[])), sid_69 = sid_2168 &
attacker:host(pk(skB[])) ->
```

```
attacker:sencrypt(secretANa[],Na[sign((v_2167,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_2168])
Rule 26:
attacker:encrypt((Na[sign((v_2170,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_2171],v_2172),pk(skA[])) &
attacker:sign((v_2170,host(pk(skB[]))),skS[]) &
begin:beginBparam(host(pk(skB[]))), m_32 =
encrypt((Na[sign((v_2170,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_21
71],v_2172),pk(skA[])), ms_30 = sign((v_2170,host(pk(skB[]))),skS[]),
hostX_29 = host(pk(skB[])), sid_69 = sid_2171 &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_2172)
Rule 27: attacker:encrypt((v_2179,v_2180),pk(skB[])) ->
attacker:(host(pk(skB[])),v_2180)
Rule 28: attacker:sign((v_2187,v_2188),skS[]) &
attacker:encrypt((v_2189,v_2188),pk(skB[])) ->
attacker:encrypt((v_2189,Nb[sign((v_2187,v_2188),skS[]),encrypt((v_2189
,v_2188),pk(skB[])),sid_2190]),v_2187)
Rule 29:
attacker:encrypt(Nb[sign((v_2194,host(pk(skA[]))),skS[]),encrypt((v_219
5,host(pk(skA[]))),pk(skB[])),sid_2196],pk(skB[])) &
attacker:sign((v_2194,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2195,host(pk(skA[]))),pk(skB[])) ->
end:sid_2196,endBparam(host(pk(skB[])))
Rule 30:
attacker:encrypt(Nb[sign((v_2197,host(pk(skA[]))),skS[]),encrypt((v_219
8,host(pk(skA[]))),pk(skB[])),sid_2199],pk(skB[])) &
attacker:sign((v_2197,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2198,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_2198)
Rule 31:
attacker:encrypt(Nb[sign((v_2200,host(pk(skA[]))),skS[]),encrypt((v_220
1,host(pk(skA[]))),pk(skB[])),sid_2202],pk(skB[])) &
attacker:sign((v_2200,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2201,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_2200,host(pk(skA[]))),skS[]),e
ncrypt((v_2201,host(pk(skA[]))),pk(skB[])),sid_2202])
```

```
Rule 32: attacker:(v_2208,host(x_2209)) ->
attacker:sign((x_2209,host(x_2209)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query evinj:endBparam(x_2116) ==> evinj:beginBparam(x_2116)
goal reachable: begin:beginBparam(host(pk(y_2740))), m_32 =
encrypt((Na[sign((pk(y_2740),host(pk(y_2740))),skS[]),host(pk(y_2740)),
sid_2741],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((
pk(y_2740),host(pk(y_2740))),skS[]),host(pk(y_2740)),sid_2741],host(pk(
skA[]))),pk(skB[])),endsid_2742]),pk(skA[])), ms_30 =
sign((pk(y_2740),host(pk(y_2740))),skS[]), hostX_29 = host(pk(y_2740)),
sid_69 = sid_2741 & begin:beginBparam(host(pk(y_2740))), ms_30 =
sign((pk(y_2740),host(pk(y_2740))),skS[]), hostX_29 = host(pk(y_2740)),
sid_69 = sid_2741 & attacker:y_2740 ->
end:endsid_2742,endBparam(host(pk(skB[])))
rule 29 end:endsid_2827,endBparam(host(pk(skB[])))
  rule 4
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821)),sid_2818],h
ost(pk(skA[]))),pk(skB[])),endsid_2827],pk(skB[]))
    rule 8
attacker:Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((p
k(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821)),sid_2818],host(pk(s
kA[]))),pk(skB[])),endsid_2827]
      rule 24
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821)),sid_2818],h
ost(pk(skA[]))),pk(skB[])),endsid_2827],pk(y_2821))
        rule 28
attacker:encrypt((Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(
y_2821)),sid_2818],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((
Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821)),sid_2818]
,host(pk(skA[]))),pk(skB[])),endsid_2827]),pk(skA[]))
          duplicate attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
```

```
          duplicate
attacker:encrypt((Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(
y_2821)),sid_2818],host(pk(skA[]))),pk(skB[]))
        duplicate attacker:sign((pk(y_2821),host(pk(y_2821))),skS[])
        hypothesis begin:beginBparam(host(pk(y_2821))), m_32 =
encrypt((Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821)),
sid_2818],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((
pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821)),sid_2818],host(pk(
skA[]))),pk(skB[])),endsid_2827]),pk(skA[])), ms_30 =
sign((pk(y_2821),host(pk(y_2821))),skS[]), hostX_29 = host(pk(y_2821)),
sid_69 = sid_2818
        duplicate attacker:host(pk(y_2821))
      hypothesis attacker:y_2821
    duplicate attacker:pk(skB[])
  rule 32 attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
    2-tuple attacker:(v_2756,host(pk(skA[])))
      any attacker:v_2756
      duplicate attacker:host(pk(skA[]))
  rule 4
attacker:encrypt((Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(
y_2821)),sid_2818],host(pk(skA[]))),pk(skB[]))
    2-tuple
attacker:(Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821))
,sid_2818],host(pk(skA[])))
      0-th
attacker:Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821)),
sid_2818]
        rule 8
attacker:(Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(y_2821))
,sid_2818],host(pk(skA[])))
          rule 23
attacker:encrypt((Na[sign((pk(y_2821),host(pk(y_2821))),skS[]),host(pk(
y_2821)),sid_2818],host(pk(skA[]))),pk(y_2821))
            rule 32 attacker:sign((pk(y_2821),host(pk(y_2821))),skS[])
              2-tuple attacker:(v_2764,host(pk(y_2821)))
                any attacker:v_2764
                duplicate attacker:host(pk(y_2821))
```

```
            hypothesis begin:beginBparam(host(pk(y_2821))), ms_30 =
sign((pk(y_2821),host(pk(y_2821))),skS[]), hostX_29 = host(pk(y_2821)),
sid_69 = sid_2818
            rule 6 attacker:host(pk(y_2821))
              rule 7 attacker:pk(y_2821)
                hypothesis attacker:y_2821
          hypothesis attacker:y_2821
      rule 6 attacker:host(pk(skA[]))
        rule 17 attacker:pk(skA[])
    rule 18 attacker:pk(skB[])

A more detailed output of the traces is available with
  param traceDisplay = long.
Goal of the attack :
end:a_13[],endBparam(host(pk(skB_20[])))

out(c, pk(skA_23))

out(c, pk(skB_20))

out(c, pk(skS_21))

out(c, host(pk(skA_23)))

out(c, host(pk(skB_20)))

in(c, host(pk(a_14)))

event(beginBparam(host(pk(a_14))))

out(c, (host(pk(skA_23)),host(pk(a_14))))

in(c, (a_16,host(pk(skA_23))))

out(c, sign((pk(skA_23),host(pk(skA_23))),skS_21))

in(c, (a_18,host(pk(a_14))))
```

```
out(c, sign((pk(a_14),host(pk(a_14))),skS_21))

in(c, sign((pk(a_14),host(pk(a_14))),skS_21))

out(c, encrypt((Na_22,host(pk(skA_23))),pk(a_14)))

in(c, encrypt((Na_22,host(pk(skA_23))),pk(skB_20)))

event(beginAparam(host(pk(skA_23))))

out(c, (host(pk(skB_20)),host(pk(skA_23))))

in(c, sign((pk(skA_23),host(pk(skA_23))),skS_21))

event(beginAfull(Na_22,host(pk(skA_23)),host(pk(skB_20)),pk(skB_20),pk(
skA_23),Nb_24))

out(c, encrypt((Na_22,Nb_24),pk(skA_23)))

in(c, encrypt((Na_22,Nb_24),pk(skA_23)))

event(beginBfull(Na_22,host(pk(skA_23)),host(pk(a_14)),pk(a_14),pk(skA_
23),Nb_24))

out(c, encrypt(Nb_24,pk(a_14)))

in(c, encrypt(Nb_24,pk(skB_20)))

event(endBparam(host(pk(skB_20))))

An attack has been found.
RESULT evinj:endBparam(x_2116) ==> evinj:beginBparam(x_2116) is false.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_3003,v_3003
Rule 1: attacker:sign(x_3005,y_3006) -> attacker:x_3005
```

```
Rule 2: attacker:sencrypt(x_3007,y_3008) & attacker:y_3008 ->
attacker:x_3007
Rule 3: attacker:sign(x_3009,y_3010) & attacker:pk(y_3010) ->
attacker:x_3009
Rule 4: attacker:v_3012 & attacker:v_3011 ->
attacker:encrypt(v_3012,v_3011)
Rule 5: attacker:v_3014 & attacker:v_3013 ->
attacker:sign(v_3014,v_3013)
Rule 6: attacker:v_3015 -> attacker:host(v_3015)
Rule 7: attacker:v_3016 -> attacker:pk(v_3016)
Rule 8: attacker:encrypt(x_3017,pk(y_3018)) & attacker:y_3018 ->
attacker:x_3017
Rule 9: attacker:v_3020 & attacker:v_3019 ->
attacker:sencrypt(v_3020,v_3019)
Rule 10: attacker:v_3022 & attacker:v_3021 -> attacker:(v_3022,v_3021)
Rule 11: attacker:(v_3024,v_3023) -> attacker:v_3024
Rule 12: attacker:(v_3026,v_3025) -> attacker:v_3025
Rule 13: mess:v_3028,v_3027 & attacker:v_3028 -> attacker:v_3027
Rule 14: attacker:v_3030 & attacker:v_3029 -> mess:v_3030,v_3029
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_3031]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: attacker:v_3034 -> attacker:(host(pk(skA[])),v_3034)
Rule 23: attacker:sign((v_3041,v_3042),skS[]) & attacker:v_3042 ->
attacker:encrypt((Na[sign((v_3041,v_3042),skS[]),v_3042,sid_3043],host(
pk(skA[]))),v_3041)
Rule 24:
attacker:encrypt((Na[sign((v_3049,v_3050),skS[]),v_3050,sid_3051],v_305
2),pk(skA[])) & attacker:sign((v_3049,v_3050),skS[]) & attacker:v_3050
-> attacker:encrypt(v_3052,v_3049)
Rule 25:
attacker:encrypt((Na[sign((v_3053,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3054],v_3055),pk(skA[])) &
```

```
attacker:sign((v_3053,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
end:endAfull(Na[sign((v_3053,host(pk(skB[]))),skS[]),host(pk(skB[])),si
d_3054],host(pk(skA[])),host(pk(skB[])),v_3053,pk(skA[]),v_3055)
Rule 26:
attacker:encrypt((Na[sign((v_3056,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3057],v_3058),pk(skA[])) &
attacker:sign((v_3056,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_3056,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_3057])
Rule 27:
attacker:encrypt((Na[sign((v_3059,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3060],v_3061),pk(skA[])) &
attacker:sign((v_3059,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_3061)
Rule 28: attacker:encrypt((v_3068,v_3069),pk(skB[])) ->
attacker:(host(pk(skB[])),v_3069)
Rule 29:
begin:beginAfull(v_3076,v_3077,host(pk(skB[])),pk(skB[]),v_3078,Nb[sign
((v_3078,v_3077),skS[]),encrypt((v_3076,v_3077),pk(skB[])),sid_3079]) &
attacker:sign((v_3078,v_3077),skS[]) &
attacker:encrypt((v_3076,v_3077),pk(skB[])) ->
attacker:encrypt((v_3076,Nb[sign((v_3078,v_3077),skS[]),encrypt((v_3076
,v_3077),pk(skB[])),sid_3079]),v_3078)
Rule 30:
attacker:encrypt(Nb[sign((v_3083,host(pk(skA[]))),skS[]),encrypt((v_308
4,host(pk(skA[]))),pk(skB[])),sid_3085],pk(skB[])) &
begin:beginAfull(v_3084,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_308
3,Nb[sign((v_3083,host(pk(skA[]))),skS[]),encrypt((v_3084,host(pk(skA[]
))),pk(skB[])),sid_3085]) &
attacker:sign((v_3083,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_3084,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_3084)
Rule 31:
attacker:encrypt(Nb[sign((v_3086,host(pk(skA[]))),skS[]),encrypt((v_308
7,host(pk(skA[]))),pk(skB[])),sid_3088],pk(skB[])) &
```

```
begin:beginAfull(v_3087,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_308
6,Nb[sign((v_3086,host(pk(skA[]))),skS[]),encrypt((v_3087,host(pk(skA[]
))),pk(skB[])),sid_3088]) &
attacker:sign((v_3086,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_3087,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_3086,host(pk(skA[]))),skS[]),e
ncrypt((v_3087,host(pk(skA[]))),pk(skB[])),sid_3088])
Rule 32: attacker:(v_3094,host(x_3095)) ->
attacker:sign((x_3095,host(x_3095)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query
ev:endAfull(x1_2997,x2_2998,x3_2999,x4_3000,x5_3001,x6_3002) ==>
ev:beginAfull(x1_2997,x2_2998,x3_2999,x4_3000,x5_3001,x6_3002)
goal reachable:
begin:beginAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB
[])),sid_3575],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[s
ign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),host
(pk(skB[]))),skS[]),host(pk(skB[])),sid_3575],host(pk(skA[]))),pk(skB[]
)),sid_3576]) ->
end:endAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[]))
,sid_3575],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[sign(
(pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),host(pk(
skB[]))),skS[]),host(pk(skB[])),sid_3575],host(pk(skA[]))),pk(skB[])),s
id_3576])
RESULT ev:endAfull(x1_2997,x2_2998,x3_2999,x4_3000,x5_3001,x6_3002) ==>
ev:beginAfull(x1_2997,x2_2998,x3_2999,x4_3000,x5_3001,x6_3002) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_3584,v_3584
Rule 1: attacker:sign(x_3586,y_3587) -> attacker:x_3586
Rule 2: attacker:sencrypt(x_3588,y_3589) & attacker:y_3589 ->
attacker:x_3588
Rule 3: attacker:sign(x_3590,y_3591) & attacker:pk(y_3591) ->
attacker:x_3590
```

```
Rule 4: attacker:v_3593 & attacker:v_3592 ->
attacker:encrypt(v_3593,v_3592)
Rule 5: attacker:v_3595 & attacker:v_3594 ->
attacker:sign(v_3595,v_3594)
Rule 6: attacker:v_3596 -> attacker:host(v_3596)
Rule 7: attacker:v_3597 -> attacker:pk(v_3597)
Rule 8: attacker:encrypt(x_3598,pk(y_3599)) & attacker:y_3599 ->
attacker:x_3598
Rule 9: attacker:v_3601 & attacker:v_3600 ->
attacker:sencrypt(v_3601,v_3600)
Rule 10: attacker:v_3603 & attacker:v_3602 -> attacker:(v_3603,v_3602)
Rule 11: attacker:(v_3605,v_3604) -> attacker:v_3605
Rule 12: attacker:(v_3607,v_3606) -> attacker:v_3606
Rule 13: mess:v_3609,v_3608 & attacker:v_3609 -> attacker:v_3608
Rule 14: attacker:v_3611 & attacker:v_3610 -> mess:v_3611,v_3610
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_3612]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: attacker:v_3615 -> attacker:(host(pk(skA[])),v_3615)
Rule 23: attacker:sign((v_3622,v_3623),skS[]) & attacker:v_3623 ->
attacker:encrypt((Na[sign((v_3622,v_3623),skS[]),v_3623,sid_3624],host(
pk(skA[]))),v_3622)
Rule 24:
attacker:encrypt((Na[sign((v_3630,v_3631),skS[]),v_3631,sid_3632],v_363
3),pk(skA[])) & attacker:sign((v_3630,v_3631),skS[]) & attacker:v_3631
-> attacker:encrypt(v_3633,v_3630)
Rule 25:
attacker:encrypt((Na[sign((v_3634,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3635],v_3636),pk(skA[])) &
attacker:sign((v_3634,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> end:endAparam(host(pk(skA[])))
Rule 26:
attacker:encrypt((Na[sign((v_3637,host(pk(skB[]))),skS[]),host(pk(skB[]
```

```
)),sid_3638],v_3639),pk(skA[])) &
attacker:sign((v_3637,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_3637,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_3638])
Rule 27:
attacker:encrypt((Na[sign((v_3640,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3641],v_3642),pk(skA[])) &
attacker:sign((v_3640,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_3642)
Rule 28: begin:beginAparam(v_3649) &
attacker:encrypt((v_3650,v_3649),pk(skB[])) ->
attacker:(host(pk(skB[])),v_3649)
Rule 29: attacker:sign((v_3657,v_3658),skS[]) &
begin:beginAparam(v_3658) & attacker:encrypt((v_3659,v_3658),pk(skB[]))
->
attacker:encrypt((v_3659,Nb[sign((v_3657,v_3658),skS[]),encrypt((v_3659
,v_3658),pk(skB[])),sid_3660]),v_3657)
Rule 30:
attacker:encrypt(Nb[sign((v_3664,host(pk(skA[]))),skS[]),encrypt((v_366
5,host(pk(skA[]))),pk(skB[])),sid_3666],pk(skB[])) &
attacker:sign((v_3664,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))) &
attacker:encrypt((v_3665,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_3665)
Rule 31:
attacker:encrypt(Nb[sign((v_3667,host(pk(skA[]))),skS[]),encrypt((v_366
8,host(pk(skA[]))),pk(skB[])),sid_3669],pk(skB[])) &
attacker:sign((v_3667,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))) &
attacker:encrypt((v_3668,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_3667,host(pk(skA[]))),skS[]),e
ncrypt((v_3668,host(pk(skA[]))),pk(skB[])),sid_3669])
Rule 32: attacker:(v_3675,host(x_3676)) ->
attacker:sign((x_3676,host(x_3676)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
```

```
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query ev:endAparam(x_3583) ==> ev:beginAparam(x_3583)
goal reachable: begin:beginAparam(host(pk(skA[]))) ->
end:endAparam(host(pk(skA[])))
RESULT ev:endAparam(x_3583) ==> ev:beginAparam(x_3583) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_4093,v_4093
Rule 1: attacker:sign(x_4095,y_4096) -> attacker:x_4095
Rule 2: attacker:sencrypt(x_4097,y_4098) & attacker:y_4098 ->
attacker:x_4097
Rule 3: attacker:sign(x_4099,y_4100) & attacker:pk(y_4100) ->
attacker:x_4099
Rule 4: attacker:v_4102 & attacker:v_4101 ->
attacker:encrypt(v_4102,v_4101)
Rule 5: attacker:v_4104 & attacker:v_4103 ->
attacker:sign(v_4104,v_4103)
Rule 6: attacker:v_4105 -> attacker:host(v_4105)
Rule 7: attacker:v_4106 -> attacker:pk(v_4106)
Rule 8: attacker:encrypt(x_4107,pk(y_4108)) & attacker:y_4108 ->
attacker:x_4107
Rule 9: attacker:v_4110 & attacker:v_4109 ->
attacker:sencrypt(v_4110,v_4109)
Rule 10: attacker:v_4112 & attacker:v_4111 -> attacker:(v_4112,v_4111)
Rule 11: attacker:(v_4114,v_4113) -> attacker:v_4114
Rule 12: attacker:(v_4116,v_4115) -> attacker:v_4115
Rule 13: mess:v_4118,v_4117 & attacker:v_4118 -> attacker:v_4117
Rule 14: attacker:v_4120 & attacker:v_4119 -> mess:v_4120,v_4119
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_4121]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: attacker:v_4124 -> attacker:(host(pk(skA[])),v_4124)
```

```
Rule 23: attacker:sign((v_4131,v_4132),skS[]) & attacker:v_4132 ->
attacker:encrypt((Na[sign((v_4131,v_4132),skS[]),v_4132,sid_4133],host(
pk(skA[]))),v_4131)
Rule 24:
begin:beginBfull(Na[sign((v_4139,v_4140),skS[]),v_4140,sid_4141],host(p
k(skA[])),v_4140,v_4139,pk(skA[]),v_4142) &
attacker:encrypt((Na[sign((v_4139,v_4140),skS[]),v_4140,sid_4141],v_414
2),pk(skA[])) & attacker:sign((v_4139,v_4140),skS[]) & attacker:v_4140
-> attacker:encrypt(v_4142,v_4139)
Rule 25:
begin:beginBfull(Na[sign((v_4143,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_4144],host(pk(skA[])),host(pk(skB[])),v_4143,pk(skA[]),v_4145) &
attacker:encrypt((Na[sign((v_4143,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_4144],v_4145),pk(skA[])) &
attacker:sign((v_4143,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_4143,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_4144])
Rule 26:
begin:beginBfull(Na[sign((v_4146,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_4147],host(pk(skA[])),host(pk(skB[])),v_4146,pk(skA[]),v_4148) &
attacker:encrypt((Na[sign((v_4146,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_4147],v_4148),pk(skA[])) &
attacker:sign((v_4146,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_4148)
Rule 27: attacker:encrypt((v_4155,v_4156),pk(skB[])) ->
attacker:(host(pk(skB[])),v_4156)
Rule 28: attacker:sign((v_4163,v_4164),skS[]) &
attacker:encrypt((v_4165,v_4164),pk(skB[])) ->
attacker:encrypt((v_4165,Nb[sign((v_4163,v_4164),skS[]),encrypt((v_4165
,v_4164),pk(skB[])),sid_4166]),v_4163)
Rule 29:
attacker:encrypt(Nb[sign((v_4170,host(pk(skA[]))),skS[]),encrypt((v_417
1,host(pk(skA[]))),pk(skB[])),sid_4172],pk(skB[])) &
attacker:sign((v_4170,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_4171,host(pk(skA[]))),pk(skB[])) ->
end:endBfull(v_4171,host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[])
```

```
,Nb[sign((v_4170,host(pk(skA[]))),skS[]),encrypt((v_4171,host(pk(skA[])
)),pk(skB[])),sid_4172])
Rule 30:
attacker:encrypt(Nb[sign((v_4173,host(pk(skA[]))),skS[]),encrypt((v_417
4,host(pk(skA[]))),pk(skB[])),sid_4175],pk(skB[])) &
attacker:sign((v_4173,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_4174,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_4174)
Rule 31:
attacker:encrypt(Nb[sign((v_4176,host(pk(skA[]))),skS[]),encrypt((v_417
7,host(pk(skA[]))),pk(skB[])),sid_4178],pk(skB[])) &
attacker:sign((v_4176,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_4177,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_4176,host(pk(skA[]))),skS[]),e
ncrypt((v_4177,host(pk(skA[]))),pk(skB[])),sid_4178])
Rule 32: attacker:(v_4184,host(x_4185)) ->
attacker:sign((x_4185,host(x_4185)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query
ev:endBfull(x1_4087,x2_4088,x3_4089,x4_4090,x5_4091,x6_4092) ==>
ev:beginBfull(x1_4087,x2_4088,x3_4089,x4_4090,x5_4091,x6_4092)
goal reachable:
begin:beginBfull(Na[sign((pk(y_4708),host(pk(y_4708))),skS[]),host(pk(y
_4708)),sid_4709],host(pk(skA[])),host(pk(y_4708)),pk(y_4708),pk(skA[])
,Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(y_4708
),host(pk(y_4708))),skS[]),host(pk(y_4708)),sid_4709],host(pk(skA[]))),
pk(skB[])),sid_4710]) & attacker:y_4708 ->
end:endBfull(Na[sign((pk(y_4708),host(pk(y_4708))),skS[]),host(pk(y_470
8)),sid_4709],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[si
gn((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(y_4708),host
(pk(y_4708))),skS[]),host(pk(y_4708)),sid_4709],host(pk(skA[]))),pk(skB
[])),sid_4710])
rule 29
end:endBfull(Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y_479
```

```
2)),sid_4789],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[si
gn((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(y_4792),host
(pk(y_4792))),skS[]),host(pk(y_4792)),sid_4789],host(pk(skA[]))),pk(skB
[])),sid_4797])
  rule 4
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y_4792)),sid_4789],h
ost(pk(skA[]))),pk(skB[])),sid_4797],pk(skB[]))
    rule 8
attacker:Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((p
k(y_4792),host(pk(y_4792))),skS[]),host(pk(y_4792)),sid_4789],host(pk(s
kA[]))),pk(skB[])),sid_4797]
      rule 24
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y_4792)),sid_4789],h
ost(pk(skA[]))),pk(skB[])),sid_4797],pk(y_4792))
        hypothesis
begin:beginBfull(Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y
_4792)),sid_4789],host(pk(skA[])),host(pk(y_4792)),pk(y_4792),pk(skA[])
,Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(y_4792
),host(pk(y_4792))),skS[]),host(pk(y_4792)),sid_4789],host(pk(skA[]))),
pk(skB[])),sid_4797])
        rule 28
attacker:encrypt((Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(
y_4792)),sid_4789],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((
Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y_4792)),sid_4789]
,host(pk(skA[]))),pk(skB[])),sid_4797]),pk(skA[]))
          duplicate attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
          duplicate
attacker:encrypt((Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(
y_4792)),sid_4789],host(pk(skA[]))),pk(skB[]))
        duplicate attacker:sign((pk(y_4792),host(pk(y_4792))),skS[])
        duplicate attacker:host(pk(y_4792))
      hypothesis attacker:y_4792
    duplicate attacker:pk(skB[])
  rule 32 attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
    2-tuple attacker:(v_4727,host(pk(skA[])))
```

```
      any attacker:v_4727
      duplicate attacker:host(pk(skA[]))
  rule 4
attacker:encrypt((Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(
y_4792)),sid_4789],host(pk(skA[]))),pk(skB[]))
    2-tuple
attacker:(Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y_4792))
,sid_4789],host(pk(skA[])))
      0-th
attacker:Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y_4792)),
sid_4789]
        rule 8
attacker:(Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(y_4792))
,sid_4789],host(pk(skA[])))
          rule 23
attacker:encrypt((Na[sign((pk(y_4792),host(pk(y_4792))),skS[]),host(pk(
y_4792)),sid_4789],host(pk(skA[]))),pk(y_4792))
            rule 32 attacker:sign((pk(y_4792),host(pk(y_4792))),skS[])
              2-tuple attacker:(v_4735,host(pk(y_4792)))
                any attacker:v_4735
                duplicate attacker:host(pk(y_4792))
            rule 6 attacker:host(pk(y_4792))
              rule 7 attacker:pk(y_4792)
                hypothesis attacker:y_4792
          hypothesis attacker:y_4792
      rule 6 attacker:host(pk(skA[]))
        rule 17 attacker:pk(skA[])
    rule 18 attacker:pk(skB[])

A more detailed output of the traces is available with
  param traceDisplay = long.
Goal of the attack :
end:endBfull(Na_33[],host(pk(skA_34[])),host(pk(skB_35[])),pk(skB_35[])
,pk(skA_34[]),Nb_36[])

out(c, pk(skA_34))
```

```
out(c, pk(skB_35))

out(c, pk(skS_32))

out(c, host(pk(skA_34)))

out(c, host(pk(skB_35)))

in(c, host(pk(a_25)))

event(beginBparam(host(pk(a_25))))

out(c, (host(pk(skA_34)),host(pk(a_25))))

in(c, (a_28,host(pk(skA_34))))

out(c, sign((pk(skA_34),host(pk(skA_34))),skS_32))

in(c, (a_30,host(pk(a_25))))

out(c, sign((pk(a_25),host(pk(a_25))),skS_32))

in(c, sign((pk(a_25),host(pk(a_25))),skS_32))

out(c, encrypt((Na_33,host(pk(skA_34))),pk(a_25)))

in(c, encrypt((Na_33,host(pk(skA_34))),pk(skB_35)))

event(beginAparam(host(pk(skA_34))))

out(c, (host(pk(skB_35)),host(pk(skA_34))))

in(c, sign((pk(skA_34),host(pk(skA_34))),skS_32))

event(beginAfull(Na_33,host(pk(skA_34)),host(pk(skB_35)),pk(skB_35),pk(
skA_34),Nb_36))
```

```
out(c, encrypt((Na_33,Nb_36),pk(skA_34)))

in(c, encrypt((Na_33,Nb_36),pk(skA_34)))

event(beginBfull(Na_33,host(pk(skA_34)),host(pk(a_25)),pk(a_25),pk(skA_
34),Nb_36))

out(c, encrypt(Nb_36,pk(a_25)))

in(c, encrypt(Nb_36,pk(skB_35)))

event(endBparam(host(pk(skB_35))))

event(endBfull(Na_33,host(pk(skA_34)),host(pk(skB_35)),pk(skB_35),pk(sk
A_34),Nb_36))

An attack has been found.
RESULT ev:endBfull(x1_4087,x2_4088,x3_4089,x4_4090,x5_4091,x6_4092) ==>
ev:beginBfull(x1_4087,x2_4088,x3_4089,x4_4090,x5_4091,x6_4092) is
false.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_4973,v_4973
Rule 1: attacker:sign(x_4975,y_4976) -> attacker:x_4975
Rule 2: attacker:sencrypt(x_4977,y_4978) & attacker:y_4978 ->
attacker:x_4977
Rule 3: attacker:sign(x_4979,y_4980) & attacker:pk(y_4980) ->
attacker:x_4979
Rule 4: attacker:v_4982 & attacker:v_4981 ->
attacker:encrypt(v_4982,v_4981)
Rule 5: attacker:v_4984 & attacker:v_4983 ->
attacker:sign(v_4984,v_4983)
Rule 6: attacker:v_4985 -> attacker:host(v_4985)
Rule 7: attacker:v_4986 -> attacker:pk(v_4986)
Rule 8: attacker:encrypt(x_4987,pk(y_4988)) & attacker:y_4988 ->
attacker:x_4987
```

```
Rule 9: attacker:v_4990 & attacker:v_4989 ->
attacker:sencrypt(v_4990,v_4989)
Rule 10: attacker:v_4992 & attacker:v_4991 -> attacker:(v_4992,v_4991)
Rule 11: attacker:(v_4994,v_4993) -> attacker:v_4994
Rule 12: attacker:(v_4996,v_4995) -> attacker:v_4995
Rule 13: mess:v_4998,v_4997 & attacker:v_4998 -> attacker:v_4997
Rule 14: attacker:v_5000 & attacker:v_4999 -> mess:v_5000,v_4999
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_5001]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
Rule 22: begin:beginBparam(v_5004) & attacker:v_5004 ->
attacker:(host(pk(skA[])),v_5004)
Rule 23: attacker:sign((v_5011,v_5012),skS[]) &
begin:beginBparam(v_5012) & attacker:v_5012 ->
attacker:encrypt((Na[sign((v_5011,v_5012),skS[]),v_5012,sid_5013],host(
pk(skA[]))),v_5011)
Rule 24:
attacker:encrypt((Na[sign((v_5019,v_5020),skS[]),v_5020,sid_5021],v_502
2),pk(skA[])) & attacker:sign((v_5019,v_5020),skS[]) &
begin:beginBparam(v_5020) & attacker:v_5020 ->
attacker:encrypt(v_5022,v_5019)
Rule 25:
attacker:encrypt((Na[sign((v_5023,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_5024],v_5025),pk(skA[])) &
attacker:sign((v_5023,host(pk(skB[]))),skS[]) &
begin:beginBparam(host(pk(skB[]))) & attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_5023,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_5024])
Rule 26:
attacker:encrypt((Na[sign((v_5026,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_5027],v_5028),pk(skA[])) &
attacker:sign((v_5026,host(pk(skB[]))),skS[]) &
```

```
begin:beginBparam(host(pk(skB[]))) & attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANb[],v_5028)
Rule 27: attacker:encrypt((v_5035,v_5036),pk(skB[])) ->
attacker:(host(pk(skB[])),v_5036)
Rule 28: attacker:sign((v_5043,v_5044),skS[]) &
attacker:encrypt((v_5045,v_5044),pk(skB[])) ->
attacker:encrypt((v_5045,Nb[sign((v_5043,v_5044),skS[]),encrypt((v_5045
,v_5044),pk(skB[])),sid_5046]),v_5043)
Rule 29:
attacker:encrypt(Nb[sign((v_5050,host(pk(skA[]))),skS[]),encrypt((v_505
1,host(pk(skA[]))),pk(skB[])),sid_5052],pk(skB[])) &
attacker:sign((v_5050,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_5051,host(pk(skA[]))),pk(skB[])) ->
end:endBparam(host(pk(skB[])))
Rule 30:
attacker:encrypt(Nb[sign((v_5053,host(pk(skA[]))),skS[]),encrypt((v_505
4,host(pk(skA[]))),pk(skB[])),sid_5055],pk(skB[])) &
attacker:sign((v_5053,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_5054,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_5054)
Rule 31:
attacker:encrypt(Nb[sign((v_5056,host(pk(skA[]))),skS[]),encrypt((v_505
7,host(pk(skA[]))),pk(skB[])),sid_5058],pk(skB[])) &
attacker:sign((v_5056,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_5057,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_5056,host(pk(skA[]))),skS[]),e
ncrypt((v_5057,host(pk(skA[]))),pk(skB[])),sid_5058])
Rule 32: attacker:(v_5064,host(x_5065)) ->
attacker:sign((x_5065,host(x_5065)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query ev:endBparam(x_4972) ==> ev:beginBparam(x_4972)
goal reachable: begin:beginBparam(host(pk(y_5489))) & attacker:y_5489 -
> end:endBparam(host(pk(skB[])))
rule 29 end:endBparam(host(pk(skB[])))
```

```
  rule 4
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(y_5561)),sid_5558],h
ost(pk(skA[]))),pk(skB[])),sid_5566],pk(skB[]))
    rule 8
attacker:Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((p
k(y_5561),host(pk(y_5561))),skS[]),host(pk(y_5561)),sid_5558],host(pk(s
kA[]))),pk(skB[])),sid_5566]
      rule 24
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(y_5561)),sid_5558],h
ost(pk(skA[]))),pk(skB[])),sid_5566],pk(y_5561))
        rule 28
attacker:encrypt((Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(
y_5561)),sid_5558],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((
Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(y_5561)),sid_5558]
,host(pk(skA[]))),pk(skB[])),sid_5566]),pk(skA[]))
          duplicate attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
          duplicate
attacker:encrypt((Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(
y_5561)),sid_5558],host(pk(skA[]))),pk(skB[]))
        duplicate attacker:sign((pk(y_5561),host(pk(y_5561))),skS[])
        hypothesis begin:beginBparam(host(pk(y_5561)))
        duplicate attacker:host(pk(y_5561))
      hypothesis attacker:y_5561
    duplicate attacker:pk(skB[])
  rule 32 attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
    2-tuple attacker:(v_5496,host(pk(skA[])))
      any attacker:v_5496
      duplicate attacker:host(pk(skA[]))
  rule 4
attacker:encrypt((Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(
y_5561)),sid_5558],host(pk(skA[]))),pk(skB[]))
    2-tuple
attacker:(Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(y_5561))
,sid_5558],host(pk(skA[])))
```

```
      0-th
attacker:Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(y_5561)),
sid_5558]
        rule 8
attacker:(Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(y_5561))
,sid_5558],host(pk(skA[])))
          rule 23
attacker:encrypt((Na[sign((pk(y_5561),host(pk(y_5561))),skS[]),host(pk(
y_5561)),sid_5558],host(pk(skA[]))),pk(y_5561))
            rule 32 attacker:sign((pk(y_5561),host(pk(y_5561))),skS[])
              2-tuple attacker:(v_5504,host(pk(y_5561)))
                any attacker:v_5504
                duplicate attacker:host(pk(y_5561))
            hypothesis begin:beginBparam(host(pk(y_5561)))
            rule 6 attacker:host(pk(y_5561))
              rule 7 attacker:pk(y_5561)
                hypothesis attacker:y_5561
          hypothesis attacker:y_5561
      rule 6 attacker:host(pk(skA[]))
        rule 17 attacker:pk(skA[])
    rule 18 attacker:pk(skB[])

A more detailed output of the traces is available with
  param traceDisplay = long.
Goal of the attack :
end:endBparam(host(pk(skB_44[])))

out(c, pk(skA_47))

out(c, pk(skB_44))

out(c, pk(skS_45))

out(c, host(pk(skA_47)))

out(c, host(pk(skB_44)))
```

```
in(c, host(pk(a_37)))

event(beginBparam(host(pk(a_37))))

out(c, (host(pk(skA_47)),host(pk(a_37))))

in(c, (a_40,host(pk(skA_47))))

out(c, sign((pk(skA_47),host(pk(skA_47))),skS_45))

in(c, (a_42,host(pk(a_37))))

out(c, sign((pk(a_37),host(pk(a_37))),skS_45))

in(c, sign((pk(a_37),host(pk(a_37))),skS_45))

out(c, encrypt((Na_46,host(pk(skA_47))),pk(a_37)))

in(c, encrypt((Na_46,host(pk(skA_47))),pk(skB_44)))

event(beginAparam(host(pk(skA_47))))

out(c, (host(pk(skB_44)),host(pk(skA_47))))

in(c, sign((pk(skA_47),host(pk(skA_47))),skS_45))

event(beginAfull(Na_46,host(pk(skA_47)),host(pk(skB_44)),pk(skB_44),pk(
skA_47),Nb_48))

out(c, encrypt((Na_46,Nb_48),pk(skA_47)))

in(c, encrypt((Na_46,Nb_48),pk(skA_47)))

event(beginBfull(Na_46,host(pk(skA_47)),host(pk(a_37)),pk(a_37),pk(skA_
47),Nb_48))

out(c, encrypt(Nb_48,pk(a_37)))
```

```
in(c, encrypt(Nb_48,pk(skB_44)))

event(endBparam(host(pk(skB_44))))

An attack has been found.
RESULT ev:endBparam(x_4972) ==> ev:beginBparam(x_4972) is false.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_5736,v_5736
Rule 1: attacker:sign(x_5738,y_5739) -> attacker:x_5738
Rule 2: attacker:sencrypt(x_5740,y_5741) & attacker:y_5741 ->
attacker:x_5740
Rule 3: attacker:sign(x_5742,y_5743) & attacker:pk(y_5743) ->
attacker:x_5742
Rule 4: attacker:v_5745 & attacker:v_5744 ->
attacker:encrypt(v_5745,v_5744)
Rule 5: attacker:v_5747 & attacker:v_5746 ->
attacker:sign(v_5747,v_5746)
Rule 6: attacker:v_5748 -> attacker:host(v_5748)
Rule 7: attacker:v_5749 -> attacker:pk(v_5749)
Rule 8: attacker:encrypt(x_5750,pk(y_5751)) & attacker:y_5751 ->
attacker:x_5750
Rule 9: attacker:v_5753 & attacker:v_5752 ->
attacker:sencrypt(v_5753,v_5752)
Rule 10: attacker:v_5755 & attacker:v_5754 -> attacker:(v_5755,v_5754)
Rule 11: attacker:(v_5757,v_5756) -> attacker:v_5757
Rule 12: attacker:(v_5759,v_5758) -> attacker:v_5758
Rule 13: mess:v_5761,v_5760 & attacker:v_5761 -> attacker:v_5760
Rule 14: attacker:v_5763 & attacker:v_5762 -> mess:v_5763,v_5762
Rule 15: attacker:c[]
Rule 16: attacker:new_name[v_5764]
Rule 17: attacker:pk(skA[])
Rule 18: attacker:pk(skB[])
Rule 19: attacker:pk(skS[])
Rule 20: attacker:host(pk(skA[]))
Rule 21: attacker:host(pk(skB[]))
```

```
Rule 22: attacker:v_5767 -> attacker:(host(pk(skA[])),v_5767)
Rule 23: attacker:sign((v_5774,v_5775),skS[]) & attacker:v_5775 ->
attacker:encrypt((Na[sign((v_5774,v_5775),skS[]),v_5775,sid_5776],host(
pk(skA[]))),v_5774)
Rule 24:
attacker:encrypt((Na[sign((v_5782,v_5783),skS[]),v_5783,sid_5784],v_578
5),pk(skA[])) & attacker:sign((v_5782,v_5783),skS[]) & attacker:v_5783
-> attacker:encrypt(v_5785,v_5782)
Rule 25:
attacker:encrypt((Na[sign((v_5786,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_5787],v_5788),pk(skA[])) &
attacker:sign((v_5786,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_5786,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_5787])
Rule 26:
attacker:encrypt((Na[sign((v_5789,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_5790],v_5791),pk(skA[])) &
attacker:sign((v_5789,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_5791)
Rule 27: attacker:encrypt((v_5798,v_5799),pk(skB[])) ->
attacker:(host(pk(skB[])),v_5799)
Rule 28: attacker:sign((v_5806,v_5807),skS[]) &
attacker:encrypt((v_5808,v_5807),pk(skB[])) ->
attacker:encrypt((v_5808,Nb[sign((v_5806,v_5807),skS[]),encrypt((v_5808
,v_5807),pk(skB[])),sid_5809]),v_5806)
Rule 29:
attacker:encrypt(Nb[sign((v_5813,host(pk(skA[]))),skS[]),encrypt((v_581
4,host(pk(skA[]))),pk(skB[])),sid_5815],pk(skB[])) &
attacker:sign((v_5813,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_5814,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_5814)
Rule 30:
attacker:encrypt(Nb[sign((v_5816,host(pk(skA[]))),skS[]),encrypt((v_581
7,host(pk(skA[]))),pk(skB[])),sid_5818],pk(skB[])) &
attacker:sign((v_5816,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_5817,host(pk(skA[]))),pk(skB[])) ->
```

```
attacker:sencrypt(secretBNb[],Nb[sign((v_5816,host(pk(skA[]))),skS[]),e
ncrypt((v_5817,host(pk(skA[]))),pk(skB[])),sid_5818])
Rule 31: attacker:(v_5824,host(x_5825)) ->
attacker:sign((x_5825,host(x_5825)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query not attacker:secretANa[]
RESULT not attacker:secretANa[] is true.
Starting query not attacker:secretANb[]
RESULT not attacker:secretANb[] is true.
Starting query not attacker:secretBNa[]
goal reachable: attacker:secretBNa[]
rule 2 attacker:secretBNa[]
  rule 29
attacker:sencrypt(secretBNa[],Na[sign((pk(y_6312),host(pk(y_6312))),skS
[]),host(pk(y_6312)),sid_6309])
    rule 4
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(y_6312)),sid_6309],h
ost(pk(skA[]))),pk(skB[])),sid_6317],pk(skB[]))
      rule 8
attacker:Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((p
k(y_6312),host(pk(y_6312))),skS[]),host(pk(y_6312)),sid_6309],host(pk(s
kA[]))),pk(skB[])),sid_6317]
        rule 24
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(y_6312)),sid_6309],h
ost(pk(skA[]))),pk(skB[])),sid_6317],pk(y_6312))
          rule 28
attacker:encrypt((Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(
y_6312)),sid_6309],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((
Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(y_6312)),sid_6309]
,host(pk(skA[]))),pk(skB[])),sid_6317]),pk(skA[]))
            duplicate attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
```

```
            duplicate
attacker:encrypt((Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(
y_6312)),sid_6309],host(pk(skA[]))),pk(skB[]))
          duplicate attacker:sign((pk(y_6312),host(pk(y_6312))),skS[])
          duplicate attacker:host(pk(y_6312))
        any attacker:y_6312
      duplicate attacker:pk(skB[])
    rule 31 attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
      2-tuple attacker:(v_6247,host(pk(skA[])))
        any attacker:v_6247
        rule 6 attacker:host(pk(skA[]))
          rule 17 attacker:pk(skA[])
    rule 4
attacker:encrypt((Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(
y_6312)),sid_6309],host(pk(skA[]))),pk(skB[]))
      duplicate
attacker:(Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(y_6312))
,sid_6309],host(pk(skA[])))
      rule 18 attacker:pk(skB[])
  0-th
attacker:Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(y_6312)),
sid_6309]
    rule 8
attacker:(Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(y_6312))
,sid_6309],host(pk(skA[])))
      rule 23
attacker:encrypt((Na[sign((pk(y_6312),host(pk(y_6312))),skS[]),host(pk(
y_6312)),sid_6309],host(pk(skA[]))),pk(y_6312))
        rule 31 attacker:sign((pk(y_6312),host(pk(y_6312))),skS[])
          2-tuple attacker:(v_6233,host(pk(y_6312)))
            any attacker:v_6233
            duplicate attacker:host(pk(y_6312))
        rule 6 attacker:host(pk(y_6312))
          rule 7 attacker:pk(y_6312)
            any attacker:y_6312
      any attacker:y_6312
```

```
A more detailed output of the traces is available with
  param traceDisplay = long.
Goal of the attack :
attacker:secretBNa[]

out(c, pk(skA_58))

out(c, pk(skB_59))

out(c, pk(skS_56))

out(c, host(pk(skA_58)))

out(c, host(pk(skB_59)))

in(c, host(pk(a_49)))

event(beginBparam(host(pk(a_49))))

out(c, (host(pk(skA_58)),host(pk(a_49))))

in(c, (a_52,host(pk(skA_58))))

out(c, sign((pk(skA_58),host(pk(skA_58))),skS_56))

in(c, (a_54,host(pk(a_49))))

out(c, sign((pk(a_49),host(pk(a_49))),skS_56))

in(c, sign((pk(a_49),host(pk(a_49))),skS_56))

out(c, encrypt((Na_57,host(pk(skA_58))),pk(a_49)))

in(c, encrypt((Na_57,host(pk(skA_58))),pk(skB_59)))

event(beginAparam(host(pk(skA_58))))
```

```
out(c, (host(pk(skB_59)),host(pk(skA_58))))

in(c, sign((pk(skA_58),host(pk(skA_58))),skS_56))

event(beginAfull(Na_57,host(pk(skA_58)),host(pk(skB_59)),pk(skB_59),pk(
skA_58),Nb_60))

out(c, encrypt((Na_57,Nb_60),pk(skA_58)))

in(c, encrypt((Na_57,Nb_60),pk(skA_58)))

event(beginBfull(Na_57,host(pk(skA_58)),host(pk(a_49)),pk(a_49),pk(skA_
58),Nb_60))

out(c, encrypt(Nb_60,pk(a_49)))

in(c, encrypt(Nb_60,pk(skB_59)))

event(endBparam(host(pk(skB_59))))

event(endBfull(Na_57,host(pk(skA_58)),host(pk(skB_59)),pk(skB_59),pk(sk
A_58),Nb_60))

out(c, sencrypt(secretBNa,Na_57))

out(c, sencrypt(secretBNb,Nb_60))

An attack has been found.
RESULT not attacker:secretBNa[] is false.
Starting query not attacker:secretBNb[]
goal reachable: attacker:secretBNb[]
rule 2 attacker:secretBNb[]
  rule 30
attacker:sencrypt(secretBNb[],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]
),encrypt((Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601)
),sid_6598],host(pk(skA[]))),pk(skB[])),sid_6606])
```

```
    rule 4
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601)),sid_6598],h
ost(pk(skA[]))),pk(skB[])),sid_6606],pk(skB[]))
      duplicate
attacker:Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((p
k(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601)),sid_6598],host(pk(s
kA[]))),pk(skB[])),sid_6606]
      duplicate attacker:pk(skB[])
    duplicate attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
    duplicate
attacker:encrypt((Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(
y_6601)),sid_6598],host(pk(skA[]))),pk(skB[]))
  rule 8
attacker:Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((p
k(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601)),sid_6598],host(pk(s
kA[]))),pk(skB[])),sid_6606]
    rule 24
attacker:encrypt(Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na
[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601)),sid_6598],h
ost(pk(skA[]))),pk(skB[])),sid_6606],pk(y_6601))
      rule 28
attacker:encrypt((Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(
y_6601)),sid_6598],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((
Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601)),sid_6598]
,host(pk(skA[]))),pk(skB[])),sid_6606]),pk(skA[]))
        rule 31 attacker:sign((pk(skA[]),host(pk(skA[]))),skS[])
          2-tuple attacker:(v_6515,host(pk(skA[])))
            any attacker:v_6515
            duplicate attacker:host(pk(skA[]))
        rule 4
attacker:encrypt((Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(
y_6601)),sid_6598],host(pk(skA[]))),pk(skB[]))
          2-tuple
attacker:(Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601))
,sid_6598],host(pk(skA[])))
```

```
            0-th
attacker:Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601)),
sid_6598]
              rule 8
attacker:(Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(y_6601))
,sid_6598],host(pk(skA[])))
                rule 23
attacker:encrypt((Na[sign((pk(y_6601),host(pk(y_6601))),skS[]),host(pk(
y_6601)),sid_6598],host(pk(skA[]))),pk(y_6601))
                  duplicate
attacker:sign((pk(y_6601),host(pk(y_6601))),skS[])
                  duplicate attacker:host(pk(y_6601))
                any attacker:y_6601
             rule 6 attacker:host(pk(skA[]))
               rule 17 attacker:pk(skA[])
           rule 18 attacker:pk(skB[])
       rule 31 attacker:sign((pk(y_6601),host(pk(y_6601))),skS[])
         2-tuple attacker:(v_6493,host(pk(y_6601)))
           any attacker:v_6493
           duplicate attacker:host(pk(y_6601))
       rule 6 attacker:host(pk(y_6601))
         rule 7 attacker:pk(y_6601)
           any attacker:y_6601
     any attacker:y_6601

A more detailed output of the traces is available with
  param traceDisplay = long.
Goal of the attack :
attacker:secretBNb[]

out(c, pk(skA_68))

out(c, pk(skB_71))

out(c, pk(skS_69))

out(c, host(pk(skA_68)))
```

```
out(c, host(pk(skB_71)))

in(c, host(pk(a_61)))

event(beginBparam(host(pk(a_61))))

out(c, (host(pk(skA_68)),host(pk(a_61))))

in(c, (a_64,host(pk(skA_68))))

out(c, sign((pk(skA_68),host(pk(skA_68))),skS_69))

in(c, (a_66,host(pk(a_61))))

out(c, sign((pk(a_61),host(pk(a_61))),skS_69))

in(c, sign((pk(a_61),host(pk(a_61))),skS_69))

out(c, encrypt((Na_70,host(pk(skA_68))),pk(a_61)))

in(c, encrypt((Na_70,host(pk(skA_68))),pk(skB_71)))

event(beginAparam(host(pk(skA_68))))

out(c, (host(pk(skB_71)),host(pk(skA_68))))

in(c, sign((pk(skA_68),host(pk(skA_68))),skS_69))

event(beginAfull(Na_70,host(pk(skA_68)),host(pk(skB_71)),pk(skB_71),pk(
skA_68),Nb_72))

out(c, encrypt((Na_70,Nb_72),pk(skA_68)))

in(c, encrypt((Na_70,Nb_72),pk(skA_68)))
```

```
event(beginBfull(Na_70,host(pk(skA_68)),host(pk(a_61)),pk(a_61),pk(skA_
68),Nb_72))

out(c, encrypt(Nb_72,pk(a_61)))

in(c, encrypt(Nb_72,pk(skB_71)))

event(endBparam(host(pk(skB_71))))

event(endBfull(Na_70,host(pk(skA_68)),host(pk(skB_71)),pk(skB_71),pk(sk
A_68),Nb_72))

out(c, sencrypt(secretBNa,Na_70))

out(c, sencrypt(secretBNb,Nb_72))

An attack has been found.
RESULT not attacker:secretBNb[] is false.
```

Listing 24: ProVerif NS Full 2 Observed Correctness Test Data

Listing 25 is the collected data for Proverif for the full version of the NSL protocol. The gray background portions of the listing show that Proverif identified that the NSL protocol is secure.

```
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_40,v_40
Rule 1: attacker:sign(x_42,y_43) -> attacker:x_42
Rule 2: attacker:sencrypt(x_44,y_45) & attacker:y_45 -> attacker:x_44
Rule 3: attacker:sign(x_46,y_47) & attacker:pk(y_47) -> attacker:x_46
Rule 4: attacker:v_49 & attacker:v_48 -> attacker:encrypt(v_49,v_48)
Rule 5: attacker:v_51 & attacker:v_50 -> attacker:sign(v_51,v_50)
Rule 6: attacker:v_52 -> attacker:host(v_52)
Rule 7: attacker:v_53 -> attacker:pk(v_53)
Rule 8: attacker:encrypt(x_54,pk(y_55)) & attacker:y_55 ->
attacker:x_54
Rule 9: attacker:v_57 & attacker:v_56 -> attacker:sencrypt(v_57,v_56)
```

```
Rule 10: attacker:v_60 & attacker:v_59 & attacker:v_58 ->
attacker:(v_60,v_59,v_58)
Rule 11: attacker:(v_63,v_62,v_61) -> attacker:v_63
Rule 12: attacker:(v_66,v_65,v_64) -> attacker:v_65
Rule 13: attacker:(v_69,v_68,v_67) -> attacker:v_67
Rule 14: attacker:v_71 & attacker:v_70 -> attacker:(v_71,v_70)
Rule 15: attacker:(v_73,v_72) -> attacker:v_73
Rule 16: attacker:(v_75,v_74) -> attacker:v_74
Rule 17: mess:v_77,v_76 & attacker:v_77 -> attacker:v_76
Rule 18: attacker:v_79 & attacker:v_78 -> mess:v_79,v_78
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_80]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: attacker:v_83 -> attacker:(host(pk(skA[])),v_83)
Rule 27: attacker:sign((v_90,v_91),skS[]) & attacker:v_91 ->
attacker:encrypt((Na[sign((v_90,v_91),skS[]),v_91,sid_92],host(pk(skA[]
))),v_90)
Rule 28:
attacker:encrypt((Na[sign((v_99,v_100),skS[]),v_100,sid_101],v_102,v_10
0),pk(skA[])) & attacker:sign((v_99,v_100),skS[]) & attacker:v_100 ->
attacker:encrypt(v_102,v_99)
Rule 29:
attacker:encrypt((Na[sign((v_103,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_104],v_105,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_103,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
->
end:sid_104,endAfull(Na[sign((v_103,host(pk(skB[]))),skS[]),host(pk(skB
[])),sid_104],host(pk(skA[])),host(pk(skB[])),v_103,pk(skA[]),v_105)
Rule 30:
attacker:encrypt((Na[sign((v_106,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_107],v_108,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_106,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
->
```

```
attacker:sencrypt(secretANa[],Na[sign((v_106,host(pk(skB[]))),skS[]),ho
st(pk(skB[])),sid_107])
Rule 31:
attacker:encrypt((Na[sign((v_109,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_110],v_111,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_109,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
-> attacker:sencrypt(secretANb[],v_111)
Rule 32: attacker:encrypt((v_118,v_119),pk(skB[])) ->
attacker:(host(pk(skB[])),v_119)
Rule 33:
begin:beginAfull(v_128,v_127,host(pk(skB[])),pk(skB[]),v_126,Nb[sign((v
_126,v_127),skS[]),encrypt((v_128,v_127),pk(skB[])),sid_129]), ms_26 =
sign((v_126,v_127),skS[]), m_23 = encrypt((v_128,v_127),pk(skB[])),
sid_112 = sid_129 & attacker:sign((v_126,v_127),skS[]) &
attacker:encrypt((v_128,v_127),pk(skB[])) ->
attacker:encrypt((v_128,Nb[sign((v_126,v_127),skS[]),encrypt((v_128,v_1
27),pk(skB[])),sid_129],host(pk(skB[]))),v_126)
Rule 34:
attacker:encrypt(Nb[sign((v_133,host(pk(skA[]))),skS[]),encrypt((v_134,
host(pk(skA[]))),pk(skB[])),sid_135],pk(skB[])) &
begin:beginAfull(v_134,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_133,
Nb[sign((v_133,host(pk(skA[]))),skS[]),encrypt((v_134,host(pk(skA[]))),
pk(skB[])),sid_135]), m3_28 =
encrypt(Nb[sign((v_133,host(pk(skA[]))),skS[]),encrypt((v_134,host(pk(s
kA[]))),pk(skB[])),sid_135],pk(skB[])), ms_26 =
sign((v_133,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_134,host(pk(skA[]))),pk(skB[])), sid_112 = sid_135 &
attacker:sign((v_133,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_134,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_134)
Rule 35:
attacker:encrypt(Nb[sign((v_136,host(pk(skA[]))),skS[]),encrypt((v_137,
host(pk(skA[]))),pk(skB[])),sid_138],pk(skB[])) &
begin:beginAfull(v_137,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_136,
Nb[sign((v_136,host(pk(skA[]))),skS[]),encrypt((v_137,host(pk(skA[]))),
pk(skB[])),sid_138]), m3_28 =
encrypt(Nb[sign((v_136,host(pk(skA[]))),skS[]),encrypt((v_137,host(pk(s
```

```
kA[])),pk(skB[])),sid_138],pk(skB[])), ms_26 =
sign((v_136,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_137,host(pk(skA[]))),pk(skB[])), sid_112 = sid_138 &
attacker:sign((v_136,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_137,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_136,host(pk(skA[]))),skS[]),en
crypt((v_137,host(pk(skA[]))),pk(skB[])),sid_138])
Rule 36: attacker:(v_144,host(x_145)) ->
attacker:sign((x_145,host(x_145)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query evinj:endAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39) ==>
evinj:beginAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39)
goal reachable:
begin:beginAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB
[])),endsid_505],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb
[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),ho
st(pk(skB[]))),skS[]),host(pk(skB[])),endsid_505],host(pk(skA[]))),pk(s
kB[])),sid_506]), ms_26 = sign((pk(skA[]),host(pk(skA[]))),skS[]), m_23
=
encrypt((Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),end
sid_505],host(pk(skA[]))),pk(skB[])), sid_112 = sid_506 ->
end:endsid_505,endAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host
(pk(skB[])),endsid_505],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(sk
A[]),Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(sk
B[]),host(pk(skB[]))),skS[]),host(pk(skB[])),endsid_505],host(pk(skA[])
)),pk(skB[])),sid_506])
RESULT evinj:endAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39) ==>
evinj:beginAfull(x1_34,x2_35,x3_36,x4_37,x5_38,x6_39) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_524,v_524
Rule 1: attacker:sign(x_526,y_527) -> attacker:x_526
Rule 2: attacker:sencrypt(x_528,y_529) & attacker:y_529 ->
attacker:x_528
```

```
Rule 3: attacker:sign(x_530,y_531) & attacker:pk(y_531) ->
attacker:x_530
Rule 4: attacker:v_533 & attacker:v_532 ->
attacker:encrypt(v_533,v_532)
Rule 5: attacker:v_535 & attacker:v_534 -> attacker:sign(v_535,v_534)
Rule 6: attacker:v_536 -> attacker:host(v_536)
Rule 7: attacker:v_537 -> attacker:pk(v_537)
Rule 8: attacker:encrypt(x_538,pk(y_539)) & attacker:y_539 ->
attacker:x_538
Rule 9: attacker:v_541 & attacker:v_540 ->
attacker:sencrypt(v_541,v_540)
Rule 10: attacker:v_544 & attacker:v_543 & attacker:v_542 ->
attacker:(v_544,v_543,v_542)
Rule 11: attacker:(v_547,v_546,v_545) -> attacker:v_547
Rule 12: attacker:(v_550,v_549,v_548) -> attacker:v_549
Rule 13: attacker:(v_553,v_552,v_551) -> attacker:v_551
Rule 14: attacker:v_555 & attacker:v_554 -> attacker:(v_555,v_554)
Rule 15: attacker:(v_557,v_556) -> attacker:v_557
Rule 16: attacker:(v_559,v_558) -> attacker:v_558
Rule 17: mess:v_561,v_560 & attacker:v_561 -> attacker:v_560
Rule 18: attacker:v_563 & attacker:v_562 -> mess:v_563,v_562
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_564]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: attacker:v_567 -> attacker:(host(pk(skA[])),v_567)
Rule 27: attacker:sign((v_574,v_575),skS[]) & attacker:v_575 ->
attacker:encrypt((Na[sign((v_574,v_575),skS[]),v_575,sid_576],host(pk(s
kA[]))),v_574)
Rule 28:
attacker:encrypt((Na[sign((v_583,v_584),skS[]),v_584,sid_585],v_586,v_5
84),pk(skA[])) & attacker:sign((v_583,v_584),skS[]) & attacker:v_584 ->
attacker:encrypt(v_586,v_583)
```

```
Rule 29:
attacker:encrypt((Na[sign((v_587,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_588],v_589,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_587,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
-> end:sid_588,endAparam(host(pk(skA[])))
Rule 30:
attacker:encrypt((Na[sign((v_590,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_591],v_592,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_590,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
->
attacker:sencrypt(secretANa[],Na[sign((v_590,host(pk(skB[]))),skS[]),ho
st(pk(skB[])),sid_591])
Rule 31:
attacker:encrypt((Na[sign((v_593,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_594],v_595,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_593,host(pk(skB[]))),skS[]) & attacker:host(pk(skB[]))
-> attacker:sencrypt(secretANb[],v_595)
Rule 32: begin:beginAparam(v_603), m_23 =
encrypt((v_602,v_603),pk(skB[])), sid_112 = sid_604 &
attacker:encrypt((v_602,v_603),pk(skB[])) ->
attacker:(host(pk(skB[])),v_603)
Rule 33: attacker:sign((v_610,v_611),skS[]) & begin:beginAparam(v_611),
ms_26 = sign((v_610,v_611),skS[]), m_23 =
encrypt((v_612,v_611),pk(skB[])), sid_112 = sid_613 &
attacker:encrypt((v_612,v_611),pk(skB[])) ->
attacker:encrypt((v_612,Nb[sign((v_610,v_611),skS[]),encrypt((v_612,v_6
11),pk(skB[])),sid_613],host(pk(skB[]))),v_610)
Rule 34:
attacker:encrypt(Nb[sign((v_617,host(pk(skA[]))),skS[]),encrypt((v_618,
host(pk(skA[]))),pk(skB[])),sid_619],pk(skB[])) &
attacker:sign((v_617,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))), m3_28 =
encrypt(Nb[sign((v_617,host(pk(skA[]))),skS[]),encrypt((v_618,host(pk(s
kA[]))),pk(skB[])),sid_619],pk(skB[])), ms_26 =
sign((v_617,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_618,host(pk(skA[]))),pk(skB[])), sid_112 = sid_619 &
```

```
attacker:encrypt((v_618,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_618)
Rule 35:
attacker:encrypt(Nb[sign((v_620,host(pk(skA[]))),skS[]),encrypt((v_621,
host(pk(skA[]))),pk(skB[])),sid_622],pk(skB[])) &
attacker:sign((v_620,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))), m3_28 =
encrypt(Nb[sign((v_620,host(pk(skA[]))),skS[]),encrypt((v_621,host(pk(s
kA[]))),pk(skB[])),sid_622],pk(skB[])), ms_26 =
sign((v_620,host(pk(skA[]))),skS[]), m_23 =
encrypt((v_621,host(pk(skA[]))),pk(skB[])), sid_112 = sid_622 &
attacker:encrypt((v_621,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_620,host(pk(skA[]))),skS[]),en
crypt((v_621,host(pk(skA[]))),pk(skB[])),sid_622])
Rule 36: attacker:(v_628,host(x_629)) ->
attacker:sign((x_629,host(x_629)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query evinj:endAparam(x_523) ==> evinj:beginAparam(x_523)
goal reachable: begin:beginAparam(host(pk(skA[]))), ms_26 =
sign((pk(skA[]),host(pk(skA[]))),skS[]), m_23 =
encrypt((Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),end
sid_973],host(pk(skA[]))),pk(skB[])), sid_112 = sid_974 ->
end:endsid_973,endAparam(host(pk(skA[])))
RESULT evinj:endAparam(x_523) ==> evinj:beginAparam(x_523) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_987,v_987
Rule 1: attacker:sign(x_989,y_990) -> attacker:x_989
Rule 2: attacker:sencrypt(x_991,y_992) & attacker:y_992 ->
attacker:x_991
Rule 3: attacker:sign(x_993,y_994) & attacker:pk(y_994) ->
attacker:x_993
Rule 4: attacker:v_996 & attacker:v_995 ->
attacker:encrypt(v_996,v_995)
```

```
Rule 5: attacker:v_998 & attacker:v_997 -> attacker:sign(v_998,v_997)
Rule 6: attacker:v_999 -> attacker:host(v_999)
Rule 7: attacker:v_1000 -> attacker:pk(v_1000)
Rule 8: attacker:encrypt(x_1001,pk(y_1002)) & attacker:y_1002 ->
attacker:x_1001
Rule 9: attacker:v_1004 & attacker:v_1003 ->
attacker:sencrypt(v_1004,v_1003)
Rule 10: attacker:v_1007 & attacker:v_1006 & attacker:v_1005 ->
attacker:(v_1007,v_1006,v_1005)
Rule 11: attacker:(v_1010,v_1009,v_1008) -> attacker:v_1010
Rule 12: attacker:(v_1013,v_1012,v_1011) -> attacker:v_1012
Rule 13: attacker:(v_1016,v_1015,v_1014) -> attacker:v_1014
Rule 14: attacker:v_1018 & attacker:v_1017 -> attacker:(v_1018,v_1017)
Rule 15: attacker:(v_1020,v_1019) -> attacker:v_1020
Rule 16: attacker:(v_1022,v_1021) -> attacker:v_1021
Rule 17: mess:v_1024,v_1023 & attacker:v_1024 -> attacker:v_1023
Rule 18: attacker:v_1026 & attacker:v_1025 -> mess:v_1026,v_1025
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_1027]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: attacker:v_1030 -> attacker:(host(pk(skA[])),v_1030)
Rule 27: attacker:sign((v_1037,v_1038),skS[]) & attacker:v_1038 ->
attacker:encrypt((Na[sign((v_1037,v_1038),skS[]),v_1038,sid_1039],host(
pk(skA[]))),v_1037)
Rule 28:
begin:beginBfull(Na[sign((v_1046,v_1047),skS[]),v_1047,sid_1048],host(p
k(skA[])),v_1047,v_1046,pk(skA[]),v_1049), m_32 =
encrypt((Na[sign((v_1046,v_1047),skS[]),v_1047,sid_1048],v_1049,v_1047)
,pk(skA[])), ms_30 = sign((v_1046,v_1047),skS[]), hostX_29 = v_1047,
sid_81 = sid_1048 &
attacker:encrypt((Na[sign((v_1046,v_1047),skS[]),v_1047,sid_1048],v_104
9,v_1047),pk(skA[])) & attacker:sign((v_1046,v_1047),skS[]) &
attacker:v_1047 -> attacker:encrypt(v_1049,v_1046)
```

```
Rule 29:
begin:beginBfull(Na[sign((v_1050,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_1051],host(pk(skA[])),host(pk(skB[])),v_1050,pk(skA[]),v_1052),
m_32 =
encrypt((Na[sign((v_1050,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_10
51],v_1052,host(pk(skB[]))),pk(skA[])), ms_30 =
sign((v_1050,host(pk(skB[]))),skS[]), hostX_29 = host(pk(skB[])),
sid_81 = sid_1051 &
attacker:encrypt((Na[sign((v_1050,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1051],v_1052,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_1050,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_1050,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_1051])
Rule 30:
begin:beginBfull(Na[sign((v_1053,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_1054],host(pk(skA[])),host(pk(skB[])),v_1053,pk(skA[]),v_1055),
m_32 =
encrypt((Na[sign((v_1053,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_10
54],v_1055,host(pk(skB[]))),pk(skA[])), ms_30 =
sign((v_1053,host(pk(skB[]))),skS[]), hostX_29 = host(pk(skB[])),
sid_81 = sid_1054 &
attacker:encrypt((Na[sign((v_1053,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1054],v_1055,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_1053,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_1055)
Rule 31: attacker:encrypt((v_1062,v_1063),pk(skB[])) ->
attacker:(host(pk(skB[])),v_1063)
Rule 32: attacker:sign((v_1070,v_1071),skS[]) &
attacker:encrypt((v_1072,v_1071),pk(skB[])) ->
attacker:encrypt((v_1072,Nb[sign((v_1070,v_1071),skS[]),encrypt((v_1072
,v_1071),pk(skB[])),sid_1073],host(pk(skB[]))),v_1070)
Rule 33:
attacker:encrypt(Nb[sign((v_1077,host(pk(skA[]))),skS[]),encrypt((v_107
8,host(pk(skA[]))),pk(skB[])),sid_1079],pk(skB[])) &
attacker:sign((v_1077,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1078,host(pk(skA[]))),pk(skB[])) ->
```

```
end:sid_1079,endBfull(v_1078,host(pk(skA[])),host(pk(skB[])),pk(skB[]),
pk(skA[]),Nb[sign((v_1077,host(pk(skA[]))),skS[]),encrypt((v_1078,host(
pk(skA[]))),pk(skB[])),sid_1079])
Rule 34:
attacker:encrypt(Nb[sign((v_1080,host(pk(skA[]))),skS[]),encrypt((v_108
1,host(pk(skA[]))),pk(skB[])),sid_1082],pk(skB[])) &
attacker:sign((v_1080,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1081,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_1081)
Rule 35:
attacker:encrypt(Nb[sign((v_1083,host(pk(skA[]))),skS[]),encrypt((v_108
4,host(pk(skA[]))),pk(skB[])),sid_1085],pk(skB[])) &
attacker:sign((v_1083,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1084,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_1083,host(pk(skA[]))),skS[]),e
ncrypt((v_1084,host(pk(skA[]))),pk(skB[])),sid_1085])
Rule 36: attacker:(v_1091,host(x_1092)) ->
attacker:sign((x_1092,host(x_1092)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query
evinj:endBfull(x1_981,x2_982,x3_983,x4_984,x5_985,x6_986) ==>
evinj:beginBfull(x1_981,x2_982,x3_983,x4_984,x5_985,x6_986)
goal reachable:
begin:beginBfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB
[])),sid_1431],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[s
ign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),host
(pk(skB[]))),skS[]),host(pk(skB[])),sid_1431],host(pk(skA[]))),pk(skB[]
)),endsid_1432]), m_32 =
encrypt((Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),sid
_1431],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(
skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),sid_1431],host(pk(skA[])
)),pk(skB[])),endsid_1432],host(pk(skB[]))),pk(skA[])), ms_30 =
sign((pk(skB[]),host(pk(skB[]))),skS[]), hostX_29 = host(pk(skB[])),
sid_81 = sid_1431 ->
```

```
end:endsid_1432,endBfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),hos
t(pk(skB[])),sid_1431],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA
[]),Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB
[]),host(pk(skB[]))),skS[]),host(pk(skB[])),sid_1431],host(pk(skA[]))),
pk(skB[])),endsid_1432])
RESULT evinj:endBfull(x1_981,x2_982,x3_983,x4_984,x5_985,x6_986) ==>
evinj:beginBfull(x1_981,x2_982,x3_983,x4_984,x5_985,x6_986) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_1450,v_1450
Rule 1: attacker:sign(x_1452,y_1453) -> attacker:x_1452
Rule 2: attacker:sencrypt(x_1454,y_1455) & attacker:y_1455 ->
attacker:x_1454
Rule 3: attacker:sign(x_1456,y_1457) & attacker:pk(y_1457) ->
attacker:x_1456
Rule 4: attacker:v_1459 & attacker:v_1458 ->
attacker:encrypt(v_1459,v_1458)
Rule 5: attacker:v_1461 & attacker:v_1460 ->
attacker:sign(v_1461,v_1460)
Rule 6: attacker:v_1462 -> attacker:host(v_1462)
Rule 7: attacker:v_1463 -> attacker:pk(v_1463)
Rule 8: attacker:encrypt(x_1464,pk(y_1465)) & attacker:y_1465 ->
attacker:x_1464
Rule 9: attacker:v_1467 & attacker:v_1466 ->
attacker:sencrypt(v_1467,v_1466)
Rule 10: attacker:v_1470 & attacker:v_1469 & attacker:v_1468 ->
attacker:(v_1470,v_1469,v_1468)
Rule 11: attacker:(v_1473,v_1472,v_1471) -> attacker:v_1473
Rule 12: attacker:(v_1476,v_1475,v_1474) -> attacker:v_1475
Rule 13: attacker:(v_1479,v_1478,v_1477) -> attacker:v_1477
Rule 14: attacker:v_1481 & attacker:v_1480 -> attacker:(v_1481,v_1480)
Rule 15: attacker:(v_1483,v_1482) -> attacker:v_1483
Rule 16: attacker:(v_1485,v_1484) -> attacker:v_1484
Rule 17: mess:v_1487,v_1486 & attacker:v_1487 -> attacker:v_1486
Rule 18: attacker:v_1489 & attacker:v_1488 -> mess:v_1489,v_1488
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_1490]
```

```
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: begin:beginBparam(v_1493), hostX_29 = v_1493, sid_81 =
sid_1494 & attacker:v_1493 -> attacker:(host(pk(skA[])),v_1493)
Rule 27: attacker:sign((v_1500,v_1501),skS[]) &
begin:beginBparam(v_1501), ms_30 = sign((v_1500,v_1501),skS[]),
hostX_29 = v_1501, sid_81 = sid_1502 & attacker:v_1501 ->
attacker:encrypt((Na[sign((v_1500,v_1501),skS[]),v_1501,sid_1502],host(
pk(skA[]))),v_1500)
Rule 28:
attacker:encrypt((Na[sign((v_1509,v_1510),skS[]),v_1510,sid_1511],v_151
2,v_1510),pk(skA[])) & attacker:sign((v_1509,v_1510),skS[]) &
begin:beginBparam(v_1510), m_32 =
encrypt((Na[sign((v_1509,v_1510),skS[]),v_1510,sid_1511],v_1512,v_1510)
,pk(skA[])), ms_30 = sign((v_1509,v_1510),skS[]), hostX_29 = v_1510,
sid_81 = sid_1511 & attacker:v_1510 -> attacker:encrypt(v_1512,v_1509)
Rule 29:
attacker:encrypt((Na[sign((v_1513,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1514],v_1515,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_1513,host(pk(skB[]))),skS[]) &
begin:beginBparam(host(pk(skB[]))), m_32 =
encrypt((Na[sign((v_1513,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_15
14],v_1515,host(pk(skB[]))),pk(skA[])), ms_30 =
sign((v_1513,host(pk(skB[]))),skS[]), hostX_29 = host(pk(skB[])),
sid_81 = sid_1514 & attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_1513,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_1514])
Rule 30:
attacker:encrypt((Na[sign((v_1516,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1517],v_1518,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_1516,host(pk(skB[]))),skS[]) &
begin:beginBparam(host(pk(skB[]))), m_32 =
encrypt((Na[sign((v_1516,host(pk(skB[]))),skS[]),host(pk(skB[])),sid_15
17],v_1518,host(pk(skB[]))),pk(skA[])), ms_30 =
```

```
sign((v_1516,host(pk(skB[]))),skS[]), hostX_29 = host(pk(skB[])),
sid_81 = sid_1517 & attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANb[],v_1518)
Rule 31: attacker:encrypt((v_1525,v_1526),pk(skB[])) ->
attacker:(host(pk(skB[])),v_1526)
Rule 32: attacker:sign((v_1533,v_1534),skS[]) &
attacker:encrypt((v_1535,v_1534),pk(skB[])) ->
attacker:encrypt((v_1535,Nb[sign((v_1533,v_1534),skS[]),encrypt((v_1535
,v_1534),pk(skB[])),sid_1536],host(pk(skB[]))),v_1533)
Rule 33:
attacker:encrypt(Nb[sign((v_1540,host(pk(skA[]))),skS[]),encrypt((v_154
1,host(pk(skA[]))),pk(skB[])),sid_1542],pk(skB[])) &
attacker:sign((v_1540,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1541,host(pk(skA[]))),pk(skB[])) ->
end:sid_1542,endBparam(host(pk(skB[])))
Rule 34:
attacker:encrypt(Nb[sign((v_1543,host(pk(skA[]))),skS[]),encrypt((v_154
4,host(pk(skA[]))),pk(skB[])),sid_1545],pk(skB[])) &
attacker:sign((v_1543,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1544,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_1544)
Rule 35:
attacker:encrypt(Nb[sign((v_1546,host(pk(skA[]))),skS[]),encrypt((v_154
7,host(pk(skA[]))),pk(skB[])),sid_1548],pk(skB[])) &
attacker:sign((v_1546,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_1547,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_1546,host(pk(skA[]))),skS[]),e
ncrypt((v_1547,host(pk(skA[]))),pk(skB[])),sid_1548])
Rule 36: attacker:(v_1554,host(x_1555)) ->
attacker:sign((x_1555,host(x_1555)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query evinj:endBparam(x_1449) ==> evinj:beginBparam(x_1449)
goal reachable: begin:beginBparam(host(pk(skB[]))), m_32 =
encrypt((Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),sid
```

```
_1900],Nb[sign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(
skB[]),host(pk(skB[]))),skS[]),host(pk(skB[])),sid_1900],host(pk(skA[])
)),pk(skB[])),endsid_1901],host(pk(skB[]))),pk(skA[])), ms_30 =
sign((pk(skB[]),host(pk(skB[]))),skS[]), hostX_29 = host(pk(skB[])),
sid_81 = sid_1900 & begin:beginBparam(host(pk(skB[]))), ms_30 =
sign((pk(skB[]),host(pk(skB[]))),skS[]), hostX_29 = host(pk(skB[])),
sid_81 = sid_1900 -> end:endsid_1901,endBparam(host(pk(skB[])))
RESULT evinj:endBparam(x_1449) ==> evinj:beginBparam(x_1449) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_1914,v_1914
Rule 1: attacker:sign(x_1916,y_1917) -> attacker:x_1916
Rule 2: attacker:sencrypt(x_1918,y_1919) & attacker:y_1919 ->
attacker:x_1918
Rule 3: attacker:sign(x_1920,y_1921) & attacker:pk(y_1921) ->
attacker:x_1920
Rule 4: attacker:v_1923 & attacker:v_1922 ->
attacker:encrypt(v_1923,v_1922)
Rule 5: attacker:v_1925 & attacker:v_1924 ->
attacker:sign(v_1925,v_1924)
Rule 6: attacker:v_1926 -> attacker:host(v_1926)
Rule 7: attacker:v_1927 -> attacker:pk(v_1927)
Rule 8: attacker:encrypt(x_1928,pk(y_1929)) & attacker:y_1929 ->
attacker:x_1928
Rule 9: attacker:v_1931 & attacker:v_1930 ->
attacker:sencrypt(v_1931,v_1930)
Rule 10: attacker:v_1934 & attacker:v_1933 & attacker:v_1932 ->
attacker:(v_1934,v_1933,v_1932)
Rule 11: attacker:(v_1937,v_1936,v_1935) -> attacker:v_1937
Rule 12: attacker:(v_1940,v_1939,v_1938) -> attacker:v_1939
Rule 13: attacker:(v_1943,v_1942,v_1941) -> attacker:v_1941
Rule 14: attacker:v_1945 & attacker:v_1944 -> attacker:(v_1945,v_1944)
Rule 15: attacker:(v_1947,v_1946) -> attacker:v_1947
Rule 16: attacker:(v_1949,v_1948) -> attacker:v_1948
Rule 17: mess:v_1951,v_1950 & attacker:v_1951 -> attacker:v_1950
Rule 18: attacker:v_1953 & attacker:v_1952 -> mess:v_1953,v_1952
Rule 19: attacker:c[]
```

```
Rule 20: attacker:new_name[v_1954]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: attacker:v_1957 -> attacker:(host(pk(skA[])),v_1957)
Rule 27: attacker:sign((v_1964,v_1965),skS[]) & attacker:v_1965 ->
attacker:encrypt((Na[sign((v_1964,v_1965),skS[]),v_1965,sid_1966],host(
pk(skA[]))),v_1964)
Rule 28:
attacker:encrypt((Na[sign((v_1973,v_1974),skS[]),v_1974,sid_1975],v_197
6,v_1974),pk(skA[])) & attacker:sign((v_1973,v_1974),skS[]) &
attacker:v_1974 -> attacker:encrypt(v_1976,v_1973)
Rule 29:
attacker:encrypt((Na[sign((v_1977,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1978],v_1979,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_1977,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
end:endAfull(Na[sign((v_1977,host(pk(skB[]))),skS[]),host(pk(skB[])),si
d_1978],host(pk(skA[])),host(pk(skB[])),v_1977,pk(skA[]),v_1979)
Rule 30:
attacker:encrypt((Na[sign((v_1980,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1981],v_1982,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_1980,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_1980,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_1981])
Rule 31:
attacker:encrypt((Na[sign((v_1983,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_1984],v_1985,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_1983,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_1985)
Rule 32: attacker:encrypt((v_1992,v_1993),pk(skB[])) ->
attacker:(host(pk(skB[])),v_1993)
Rule 33:
begin:beginAfull(v_2000,v_2001,host(pk(skB[])),pk(skB[]),v_2002,Nb[sign
```

```
((v_2002,v_2001),skS[]),encrypt((v_2000,v_2001),pk(skB[])),sid_2003]) &
attacker:sign((v_2002,v_2001),skS[]) &
attacker:encrypt((v_2000,v_2001),pk(skB[])) ->
attacker:encrypt((v_2000,Nb[sign((v_2002,v_2001),skS[]),encrypt((v_2000
,v_2001),pk(skB[])),sid_2003],host(pk(skB[]))),v_2002)
Rule 34:
attacker:encrypt(Nb[sign((v_2007,host(pk(skA[]))),skS[]),encrypt((v_200
8,host(pk(skA[]))),pk(skB[])),sid_2009],pk(skB[])) &
begin:beginAfull(v_2008,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_200
7,Nb[sign((v_2007,host(pk(skA[]))),skS[]),encrypt((v_2008,host(pk(skA[]
))),pk(skB[])),sid_2009]) &
attacker:sign((v_2007,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2008,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_2008)
Rule 35:
attacker:encrypt(Nb[sign((v_2010,host(pk(skA[]))),skS[]),encrypt((v_201
1,host(pk(skA[]))),pk(skB[])),sid_2012],pk(skB[])) &
begin:beginAfull(v_2011,host(pk(skA[])),host(pk(skB[])),pk(skB[]),v_201
0,Nb[sign((v_2010,host(pk(skA[]))),skS[]),encrypt((v_2011,host(pk(skA[]
))),pk(skB[])),sid_2012]) &
attacker:sign((v_2010,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2011,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_2010,host(pk(skA[]))),skS[]),e
ncrypt((v_2011,host(pk(skA[]))),pk(skB[])),sid_2012])
Rule 36: attacker:(v_2018,host(x_2019)) ->
attacker:sign((x_2019,host(x_2019)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query
ev:endAfull(x1_1908,x2_1909,x3_1910,x4_1911,x5_1912,x6_1913) ==>
ev:beginAfull(x1_1908,x2_1909,x3_1910,x4_1911,x5_1912,x6_1913)
goal reachable:
begin:beginAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB
[])),sid_2359],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[s
ign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),host
```

```
(pk(skB[]))),skS[]),host(pk(skB[])),sid_2359],host(pk(skA[]))),pk(skB[]
)),sid_2360]) ->
end:endAfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[]))
,sid_2359],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[sign(
(pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),host(pk(
skB[]))),skS[]),host(pk(skB[])),sid_2359],host(pk(skA[]))),pk(skB[])),s
id_2360])
RESULT ev:endAfull(x1_1908,x2_1909,x3_1910,x4_1911,x5_1912,x6_1913) ==>
ev:beginAfull(x1_1908,x2_1909,x3_1910,x4_1911,x5_1912,x6_1913) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_2368,v_2368
Rule 1: attacker:sign(x_2370,y_2371) -> attacker:x_2370
Rule 2: attacker:sencrypt(x_2372,y_2373) & attacker:y_2373 ->
attacker:x_2372
Rule 3: attacker:sign(x_2374,y_2375) & attacker:pk(y_2375) ->
attacker:x_2374
Rule 4: attacker:v_2377 & attacker:v_2376 ->
attacker:encrypt(v_2377,v_2376)
Rule 5: attacker:v_2379 & attacker:v_2378 ->
attacker:sign(v_2379,v_2378)
Rule 6: attacker:v_2380 -> attacker:host(v_2380)
Rule 7: attacker:v_2381 -> attacker:pk(v_2381)
Rule 8: attacker:encrypt(x_2382,pk(y_2383)) & attacker:y_2383 ->
attacker:x_2382
Rule 9: attacker:v_2385 & attacker:v_2384 ->
attacker:sencrypt(v_2385,v_2384)
Rule 10: attacker:v_2388 & attacker:v_2387 & attacker:v_2386 ->
attacker:(v_2388,v_2387,v_2386)
Rule 11: attacker:(v_2391,v_2390,v_2389) -> attacker:v_2391
Rule 12: attacker:(v_2394,v_2393,v_2392) -> attacker:v_2393
Rule 13: attacker:(v_2397,v_2396,v_2395) -> attacker:v_2395
Rule 14: attacker:v_2399 & attacker:v_2398 -> attacker:(v_2399,v_2398)
Rule 15: attacker:(v_2401,v_2400) -> attacker:v_2401
Rule 16: attacker:(v_2403,v_2402) -> attacker:v_2402
Rule 17: mess:v_2405,v_2404 & attacker:v_2405 -> attacker:v_2404
Rule 18: attacker:v_2407 & attacker:v_2406 -> mess:v_2407,v_2406
```

```
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_2408]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: attacker:v_2411 -> attacker:(host(pk(skA[])),v_2411)
Rule 27: attacker:sign((v_2418,v_2419),skS[]) & attacker:v_2419 ->
attacker:encrypt((Na[sign((v_2418,v_2419),skS[]),v_2419,sid_2420],host(
pk(skA[]))),v_2418)
Rule 28:
attacker:encrypt((Na[sign((v_2427,v_2428),skS[]),v_2428,sid_2429],v_243
0,v_2428),pk(skA[])) & attacker:sign((v_2427,v_2428),skS[]) &
attacker:v_2428 -> attacker:encrypt(v_2430,v_2427)
Rule 29:
attacker:encrypt((Na[sign((v_2431,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_2432],v_2433,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_2431,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> end:endAparam(host(pk(skA[])))
Rule 30:
attacker:encrypt((Na[sign((v_2434,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_2435],v_2436,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_2434,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_2434,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_2435])
Rule 31:
attacker:encrypt((Na[sign((v_2437,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_2438],v_2439,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_2437,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_2439)
Rule 32: begin:beginAparam(v_2446) &
attacker:encrypt((v_2447,v_2446),pk(skB[])) ->
attacker:(host(pk(skB[])),v_2446)
Rule 33: attacker:sign((v_2454,v_2455),skS[]) &
begin:beginAparam(v_2455) & attacker:encrypt((v_2456,v_2455),pk(skB[]))
```

```
->
attacker:encrypt((v_2456,Nb[sign((v_2454,v_2455),skS[]),encrypt((v_2456
,v_2455),pk(skB[])),sid_2457],host(pk(skB[]))),v_2454)
Rule 34:
attacker:encrypt(Nb[sign((v_2461,host(pk(skA[]))),skS[]),encrypt((v_246
2,host(pk(skA[]))),pk(skB[])),sid_2463],pk(skB[])) &
attacker:sign((v_2461,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))) &
attacker:encrypt((v_2462,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_2462)
Rule 35:
attacker:encrypt(Nb[sign((v_2464,host(pk(skA[]))),skS[]),encrypt((v_246
5,host(pk(skA[]))),pk(skB[])),sid_2466],pk(skB[])) &
attacker:sign((v_2464,host(pk(skA[]))),skS[]) &
begin:beginAparam(host(pk(skA[]))) &
attacker:encrypt((v_2465,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_2464,host(pk(skA[]))),skS[]),e
ncrypt((v_2465,host(pk(skA[]))),pk(skB[])),sid_2466])
Rule 36: attacker:(v_2472,host(x_2473)) ->
attacker:sign((x_2473,host(x_2473)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query ev:endAparam(x_2367) ==> ev:beginAparam(x_2367)
goal reachable: begin:beginAparam(host(pk(skA[]))) ->
end:endAparam(host(pk(skA[])))
RESULT ev:endAparam(x_2367) ==> ev:beginAparam(x_2367) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_2764,v_2764
Rule 1: attacker:sign(x_2766,y_2767) -> attacker:x_2766
Rule 2: attacker:sencrypt(x_2768,y_2769) & attacker:y_2769 ->
attacker:x_2768
Rule 3: attacker:sign(x_2770,y_2771) & attacker:pk(y_2771) ->
attacker:x_2770
```

```
Rule 4: attacker:v_2773 & attacker:v_2772 ->
attacker:encrypt(v_2773,v_2772)
Rule 5: attacker:v_2775 & attacker:v_2774 ->
attacker:sign(v_2775,v_2774)
Rule 6: attacker:v_2776 -> attacker:host(v_2776)
Rule 7: attacker:v_2777 -> attacker:pk(v_2777)
Rule 8: attacker:encrypt(x_2778,pk(y_2779)) & attacker:y_2779 ->
attacker:x_2778
Rule 9: attacker:v_2781 & attacker:v_2780 ->
attacker:sencrypt(v_2781,v_2780)
Rule 10: attacker:v_2784 & attacker:v_2783 & attacker:v_2782 ->
attacker:(v_2784,v_2783,v_2782)
Rule 11: attacker:(v_2787,v_2786,v_2785) -> attacker:v_2787
Rule 12: attacker:(v_2790,v_2789,v_2788) -> attacker:v_2789
Rule 13: attacker:(v_2793,v_2792,v_2791) -> attacker:v_2791
Rule 14: attacker:v_2795 & attacker:v_2794 -> attacker:(v_2795,v_2794)
Rule 15: attacker:(v_2797,v_2796) -> attacker:v_2797
Rule 16: attacker:(v_2799,v_2798) -> attacker:v_2798
Rule 17: mess:v_2801,v_2800 & attacker:v_2801 -> attacker:v_2800
Rule 18: attacker:v_2803 & attacker:v_2802 -> mess:v_2803,v_2802
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_2804]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: attacker:v_2807 -> attacker:(host(pk(skA[])),v_2807)
Rule 27: attacker:sign((v_2814,v_2815),skS[]) & attacker:v_2815 ->
attacker:encrypt((Na[sign((v_2814,v_2815),skS[]),v_2815,sid_2816],host(
pk(skA[]))),v_2814)
Rule 28:
begin:beginBfull(Na[sign((v_2823,v_2824),skS[]),v_2824,sid_2825],host(p
k(skA[])),v_2824,v_2823,pk(skA[]),v_2826) &
attacker:encrypt((Na[sign((v_2823,v_2824),skS[]),v_2824,sid_2825],v_282
6,v_2824),pk(skA[])) & attacker:sign((v_2823,v_2824),skS[]) &
attacker:v_2824 -> attacker:encrypt(v_2826,v_2823)
```

```
Rule 29:
begin:beginBfull(Na[sign((v_2827,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_2828],host(pk(skA[])),host(pk(skB[])),v_2827,pk(skA[]),v_2829) &
attacker:encrypt((Na[sign((v_2827,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_2828],v_2829,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_2827,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_2827,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_2828])
Rule 30:
begin:beginBfull(Na[sign((v_2830,host(pk(skB[]))),skS[]),host(pk(skB[])
),sid_2831],host(pk(skA[])),host(pk(skB[])),v_2830,pk(skA[]),v_2832) &
attacker:encrypt((Na[sign((v_2830,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_2831],v_2832,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_2830,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_2832)
Rule 31: attacker:encrypt((v_2839,v_2840),pk(skB[])) ->
attacker:(host(pk(skB[])),v_2840)
Rule 32: attacker:sign((v_2847,v_2848),skS[]) &
attacker:encrypt((v_2849,v_2848),pk(skB[])) ->
attacker:encrypt((v_2849,Nb[sign((v_2847,v_2848),skS[]),encrypt((v_2849
,v_2848),pk(skB[])),sid_2850],host(pk(skB[]))),v_2847)
Rule 33:
attacker:encrypt(Nb[sign((v_2854,host(pk(skA[]))),skS[]),encrypt((v_285
5,host(pk(skA[]))),pk(skB[])),sid_2856],pk(skB[])) &
attacker:sign((v_2854,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2855,host(pk(skA[]))),pk(skB[])) ->
end:endBfull(v_2855,host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[])
,Nb[sign((v_2854,host(pk(skA[]))),skS[]),encrypt((v_2855,host(pk(skA[])
)),pk(skB[])),sid_2856])
Rule 34:
attacker:encrypt(Nb[sign((v_2857,host(pk(skA[]))),skS[]),encrypt((v_285
8,host(pk(skA[]))),pk(skB[])),sid_2859],pk(skB[])) &
attacker:sign((v_2857,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2858,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_2858)
```

```
Rule 35:
attacker:encrypt(Nb[sign((v_2860,host(pk(skA[]))),skS[]),encrypt((v_286
1,host(pk(skA[]))),pk(skB[])),sid_2862],pk(skB[])) &
attacker:sign((v_2860,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_2861,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_2860,host(pk(skA[]))),skS[]),e
ncrypt((v_2861,host(pk(skA[]))),pk(skB[])),sid_2862])
Rule 36: attacker:(v_2868,host(x_2869)) ->
attacker:sign((x_2869,host(x_2869)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query
ev:endBfull(x1_2758,x2_2759,x3_2760,x4_2761,x5_2762,x6_2763) ==>
ev:beginBfull(x1_2758,x2_2759,x3_2760,x4_2761,x5_2762,x6_2763)
goal reachable:
begin:beginBfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB
[])),sid_3206],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[s
ign((pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),host
(pk(skB[]))),skS[]),host(pk(skB[])),sid_3206],host(pk(skA[]))),pk(skB[]
)),sid_3207]) ->
end:endBfull(Na[sign((pk(skB[]),host(pk(skB[]))),skS[]),host(pk(skB[]))
,sid_3206],host(pk(skA[])),host(pk(skB[])),pk(skB[]),pk(skA[]),Nb[sign(
(pk(skA[]),host(pk(skA[]))),skS[]),encrypt((Na[sign((pk(skB[]),host(pk(
skB[]))),skS[]),host(pk(skB[])),sid_3206],host(pk(skA[]))),pk(skB[])),s
id_3207])
RESULT ev:endBfull(x1_2758,x2_2759,x3_2760,x4_2761,x5_2762,x6_2763) ==>
ev:beginBfull(x1_2758,x2_2759,x3_2760,x4_2761,x5_2762,x6_2763) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_3215,v_3215
Rule 1: attacker:sign(x_3217,y_3218) -> attacker:x_3217
Rule 2: attacker:sencrypt(x_3219,y_3220) & attacker:y_3220 ->
attacker:x_3219
Rule 3: attacker:sign(x_3221,y_3222) & attacker:pk(y_3222) ->
attacker:x_3221
```

```
Rule 4: attacker:v_3224 & attacker:v_3223 ->
attacker:encrypt(v_3224,v_3223)
Rule 5: attacker:v_3226 & attacker:v_3225 ->
attacker:sign(v_3226,v_3225)
Rule 6: attacker:v_3227 -> attacker:host(v_3227)
Rule 7: attacker:v_3228 -> attacker:pk(v_3228)
Rule 8: attacker:encrypt(x_3229,pk(y_3230)) & attacker:y_3230 ->
attacker:x_3229
Rule 9: attacker:v_3232 & attacker:v_3231 ->
attacker:sencrypt(v_3232,v_3231)
Rule 10: attacker:v_3235 & attacker:v_3234 & attacker:v_3233 ->
attacker:(v_3235,v_3234,v_3233)
Rule 11: attacker:(v_3238,v_3237,v_3236) -> attacker:v_3238
Rule 12: attacker:(v_3241,v_3240,v_3239) -> attacker:v_3240
Rule 13: attacker:(v_3244,v_3243,v_3242) -> attacker:v_3242
Rule 14: attacker:v_3246 & attacker:v_3245 -> attacker:(v_3246,v_3245)
Rule 15: attacker:(v_3248,v_3247) -> attacker:v_3248
Rule 16: attacker:(v_3250,v_3249) -> attacker:v_3249
Rule 17: mess:v_3252,v_3251 & attacker:v_3252 -> attacker:v_3251
Rule 18: attacker:v_3254 & attacker:v_3253 -> mess:v_3254,v_3253
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_3255]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: begin:beginBparam(v_3258) & attacker:v_3258 ->
attacker:(host(pk(skA[])),v_3258)
Rule 27: attacker:sign((v_3265,v_3266),skS[]) &
begin:beginBparam(v_3266) & attacker:v_3266 ->
attacker:encrypt((Na[sign((v_3265,v_3266),skS[]),v_3266,sid_3267],host(
pk(skA[]))),v_3265)
Rule 28:
attacker:encrypt((Na[sign((v_3274,v_3275),skS[]),v_3275,sid_3276],v_327
7,v_3275),pk(skA[])) & attacker:sign((v_3274,v_3275),skS[]) &
```

```
begin:beginBparam(v_3275) & attacker:v_3275 ->
attacker:encrypt(v_3277,v_3274)
Rule 29:
attacker:encrypt((Na[sign((v_3278,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3279],v_3280,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_3278,host(pk(skB[]))),skS[]) &
begin:beginBparam(host(pk(skB[])) & attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_3278,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_3279])
Rule 30:
attacker:encrypt((Na[sign((v_3281,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3282],v_3283,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_3281,host(pk(skB[]))),skS[]) &
begin:beginBparam(host(pk(skB[])) & attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANb[],v_3283)
Rule 31: attacker:encrypt((v_3290,v_3291),pk(skB[])) ->
attacker:(host(pk(skB[])),v_3291)
Rule 32: attacker:sign((v_3298,v_3299),skS[]) &
attacker:encrypt((v_3300,v_3299),pk(skB[])) ->
attacker:encrypt((v_3300,Nb[sign((v_3298,v_3299),skS[]),encrypt((v_3300
,v_3299),pk(skB[])),sid_3301],host(pk(skB[]))),v_3298)
Rule 33:
attacker:encrypt(Nb[sign((v_3305,host(pk(skA[]))),skS[]),encrypt((v_330
6,host(pk(skA[]))),pk(skB[])),sid_3307],pk(skB[])) &
attacker:sign((v_3305,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_3306,host(pk(skA[]))),pk(skB[])) ->
end:endBparam(host(pk(skB[])))
Rule 34:
attacker:encrypt(Nb[sign((v_3308,host(pk(skA[]))),skS[]),encrypt((v_330
9,host(pk(skA[]))),pk(skB[])),sid_3310],pk(skB[])) &
attacker:sign((v_3308,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_3309,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_3309)
Rule 35:
attacker:encrypt(Nb[sign((v_3311,host(pk(skA[]))),skS[]),encrypt((v_331
2,host(pk(skA[]))),pk(skB[])),sid_3313],pk(skB[])) &
attacker:sign((v_3311,host(pk(skA[]))),skS[]) &
```

```
attacker:encrypt((v_3312,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_3311,host(pk(skA[]))),skS[]),e
ncrypt((v_3312,host(pk(skA[]))),pk(skB[])),sid_3313])
Rule 36: attacker:(v_3319,host(x_3320)) ->
attacker:sign((x_3320,host(x_3320)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query ev:endBparam(x_3214) ==> ev:beginBparam(x_3214)
goal reachable: begin:beginBparam(host(pk(skB[]))) ->
end:endBparam(host(pk(skB[])))
RESULT ev:endBparam(x_3214) ==> ev:beginBparam(x_3214) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_3606,v_3606
Rule 1: attacker:sign(x_3608,y_3609) -> attacker:x_3608
Rule 2: attacker:sencrypt(x_3610,y_3611) & attacker:y_3611 ->
attacker:x_3610
Rule 3: attacker:sign(x_3612,y_3613) & attacker:pk(y_3613) ->
attacker:x_3612
Rule 4: attacker:v_3615 & attacker:v_3614 ->
attacker:encrypt(v_3615,v_3614)
Rule 5: attacker:v_3617 & attacker:v_3616 ->
attacker:sign(v_3617,v_3616)
Rule 6: attacker:v_3618 -> attacker:host(v_3618)
Rule 7: attacker:v_3619 -> attacker:pk(v_3619)
Rule 8: attacker:encrypt(x_3620,pk(y_3621)) & attacker:y_3621 ->
attacker:x_3620
Rule 9: attacker:v_3623 & attacker:v_3622 ->
attacker:sencrypt(v_3623,v_3622)
Rule 10: attacker:v_3626 & attacker:v_3625 & attacker:v_3624 ->
attacker:(v_3626,v_3625,v_3624)
Rule 11: attacker:(v_3629,v_3628,v_3627) -> attacker:v_3629
Rule 12: attacker:(v_3632,v_3631,v_3630) -> attacker:v_3631
Rule 13: attacker:(v_3635,v_3634,v_3633) -> attacker:v_3633
Rule 14: attacker:v_3637 & attacker:v_3636 -> attacker:(v_3637,v_3636)
```

```
Rule 15: attacker:(v_3639,v_3638) -> attacker:v_3639
Rule 16: attacker:(v_3641,v_3640) -> attacker:v_3640
Rule 17: mess:v_3643,v_3642 & attacker:v_3643 -> attacker:v_3642
Rule 18: attacker:v_3645 & attacker:v_3644 -> mess:v_3645,v_3644
Rule 19: attacker:c[]
Rule 20: attacker:new_name[v_3646]
Rule 21: attacker:pk(skA[])
Rule 22: attacker:pk(skB[])
Rule 23: attacker:pk(skS[])
Rule 24: attacker:host(pk(skA[]))
Rule 25: attacker:host(pk(skB[]))
Rule 26: attacker:v_3649 -> attacker:(host(pk(skA[])),v_3649)
Rule 27: attacker:sign((v_3656,v_3657),skS[]) & attacker:v_3657 ->
attacker:encrypt((Na[sign((v_3656,v_3657),skS[]),v_3657,sid_3658],host(
pk(skA[]))),v_3656)
Rule 28:
attacker:encrypt((Na[sign((v_3665,v_3666),skS[]),v_3666,sid_3667],v_366
8,v_3666),pk(skA[])) & attacker:sign((v_3665,v_3666),skS[]) &
attacker:v_3666 -> attacker:encrypt(v_3668,v_3665)
Rule 29:
attacker:encrypt((Na[sign((v_3669,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3670],v_3671,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_3669,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) ->
attacker:sencrypt(secretANa[],Na[sign((v_3669,host(pk(skB[]))),skS[]),h
ost(pk(skB[])),sid_3670])
Rule 30:
attacker:encrypt((Na[sign((v_3672,host(pk(skB[]))),skS[]),host(pk(skB[]
)),sid_3673],v_3674,host(pk(skB[]))),pk(skA[])) &
attacker:sign((v_3672,host(pk(skB[]))),skS[]) &
attacker:host(pk(skB[])) -> attacker:sencrypt(secretANb[],v_3674)
Rule 31: attacker:encrypt((v_3681,v_3682),pk(skB[])) ->
attacker:(host(pk(skB[])),v_3682)
Rule 32: attacker:sign((v_3689,v_3690),skS[]) &
attacker:encrypt((v_3691,v_3690),pk(skB[])) ->
attacker:encrypt((v_3691,Nb[sign((v_3689,v_3690),skS[]),encrypt((v_3691
,v_3690),pk(skB[])),sid_3692],host(pk(skB[]))),v_3689)
```

```
Rule 33:
attacker:encrypt(Nb[sign((v_3696,host(pk(skA[]))),skS[]),encrypt((v_369
7,host(pk(skA[]))),pk(skB[])),sid_3698],pk(skB[])) &
attacker:sign((v_3696,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_3697,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNa[],v_3697)
Rule 34:
attacker:encrypt(Nb[sign((v_3699,host(pk(skA[]))),skS[]),encrypt((v_370
0,host(pk(skA[]))),pk(skB[])),sid_3701],pk(skB[])) &
attacker:sign((v_3699,host(pk(skA[]))),skS[]) &
attacker:encrypt((v_3700,host(pk(skA[]))),pk(skB[])) ->
attacker:sencrypt(secretBNb[],Nb[sign((v_3699,host(pk(skA[]))),skS[]),e
ncrypt((v_3700,host(pk(skA[]))),pk(skB[])),sid_3701])
Rule 35: attacker:(v_3707,host(x_3708)) ->
attacker:sign((x_3708,host(x_3708)),skS[])
Completing...
ok, secrecy assumption verified: fact unreachable attacker:skA[]
ok, secrecy assumption verified: fact unreachable attacker:skB[]
ok, secrecy assumption verified: fact unreachable attacker:skS[]
Starting query not attacker:secretANa[]
RESULT not attacker:secretANa[] is true.
Starting query not attacker:secretANb[]
RESULT not attacker:secretANb[] is true.
Starting query not attacker:secretBNa[]
RESULT not attacker:secretBNa[] is true.
Starting query not attacker:secretBNb[]
RESULT not attacker:secretBNb[] is true.
```

Listing 25: ProVerif NSL Full 2 Observed Correctness Test Data

Listing 26 is the collected data for ProVerif for the Kerberos protocol. The gray background portions of the listing show that ProVerif identified an authentication attack on the Kerberos protocol. These results were reviewed manually, and it was determined that the weakness found by ProVerif is in fact false; i.e., ProVerif was incorrect. Additionally, the gray background portions show that ProVerif verified the confidentiality characteristic of Kerberos.

```
-- Secrecy & events.
```

```
Starting rules:
Rule 0: equal:v_37,v_37
Rule 1: attacker:v_40 & attacker:v_39 -> attacker:ShareEnc(v_40,v_39)
Rule 2: attacker:ShareEnc(Msg_41,SharedKey_42) & attacker:SharedKey_42
-> attacker:Msg_41
Rule 3: attacker:v_45 & attacker:v_44 & attacker:v_43 ->
attacker:(v_45,v_44,v_43)
Rule 4: attacker:(v_48,v_47,v_46) -> attacker:v_48
Rule 5: attacker:(v_51,v_50,v_49) -> attacker:v_50
Rule 6: attacker:(v_54,v_53,v_52) -> attacker:v_52
Rule 7: attacker:v_55 -> attacker:(v_55)
Rule 8: attacker:(v_56) -> attacker:v_56
Rule 9: attacker:v_60 & attacker:v_59 & attacker:v_58 & attacker:v_57 -
> attacker:(v_60,v_59,v_58,v_57)
Rule 10: attacker:(v_64,v_63,v_62,v_61) -> attacker:v_64
Rule 11: attacker:(v_68,v_67,v_66,v_65) -> attacker:v_67
Rule 12: attacker:(v_72,v_71,v_70,v_69) -> attacker:v_70
Rule 13: attacker:(v_76,v_75,v_74,v_73) -> attacker:v_73
Rule 14: attacker:v_78 & attacker:v_77 -> attacker:(v_78,v_77)
Rule 15: attacker:(v_80,v_79) -> attacker:v_80
Rule 16: attacker:(v_82,v_81) -> attacker:v_81
Rule 17: mess:v_84,v_83 & attacker:v_84 -> attacker:v_83
Rule 18: attacker:v_86 & attacker:v_85 -> mess:v_86,v_85
Rule 19: attacker:TicServA[]
Rule 20: attacker:AppServX[]
Rule 21: attacker:UserA[]
Rule 22: attacker:chnl[]
Rule 23: attacker:new_name[v_87]
Rule 24: attacker:(UserA[],TicServA[],Nak[sid_89])
Rule 25:
attacker:((UserA[]),v_101,ShareEnc((TicServA[],v_102,Nak[sid_103],v_104
),ltkAKAS[])) ->
attacker:(AppServX[],Nat[(UserA[]),v_101,ShareEnc((TicServA[],v_102,Nak
[sid_103],v_104),ltkAKAS[]),sid_103],v_101,ShareEnc((UserA[],Tat[(UserA
[]),v_101,ShareEnc((TicServA[],v_102,Nak[sid_103],v_104),ltkAKAS[]),sid
_103]),v_102))
```

```
Rule 26:
attacker:(UserA[],v_115,ShareEnc((AppServX[],v_116,Nat[(UserA[]),v_117,
ShareEnc((TicServA[],SharedKey_118,Nak[sid_119],v_120),ltkAKAS[]),sid_1
19],v_121),SharedKey_118)) &
attacker:((UserA[]),v_117,ShareEnc((TicServA[],SharedKey_118,Nak[sid_11
9],v_120),ltkAKAS[])) ->
attacker:(v_115,ShareEnc((UserA[],Tab[v_115,ShareEnc((AppServX[],v_116,
Nat[(UserA[]),v_117,ShareEnc((TicServA[],SharedKey_118,Nak[sid_119],v_1
20),ltkAKAS[]),sid_119],v_121),SharedKey_118),(UserA[]),v_117,ShareEnc(
(TicServA[],SharedKey_118,Nak[sid_119],v_120),ltkAKAS[]),sid_119]),v_11
6))
Rule 27:
attacker:ShareEnc((Tab[v_126,ShareEnc((AppServX[],SharedKey_127,Nat[(Us
erA[]),v_128,ShareEnc((TicServA[],SharedKey_129,Nak[sid_130],v_131),ltk
AKAS[]),sid_130],v_132),SharedKey_129),(UserA[]),v_128,ShareEnc((TicSer
vA[],SharedKey_129,Nak[sid_130],v_131),ltkAKAS[]),sid_130]),SharedKey_1
27) &
attacker:(UserA[],v_126,ShareEnc((AppServX[],SharedKey_127,Nat[(UserA[]
),v_128,ShareEnc((TicServA[],SharedKey_129,Nak[sid_130],v_131),ltkAKAS[
]),sid_130],v_132),SharedKey_129)) &
attacker:((UserA[]),v_128,ShareEnc((TicServA[],SharedKey_129,Nak[sid_13
0],v_131),ltkAKAS[])) ->
attacker:ShareEnc(AttackSuccess[],SharedKey_127)
Rule 28: attacker:(v_138,v_139,v_140) ->
attacker:(v_138,ShareEnc((v_138,Kxt[(v_138,v_139,v_140),sid_141],Tk[(v_
138,v_139,v_140),sid_141]),ltkKASTGS[]),ShareEnc((v_139,Kxt[(v_138,v_13
9,v_140),sid_141],v_140,Tk[(v_138,v_139,v_140),sid_141]),ltkAKAS[]))
Rule 29:
attacker:(v_157,v_158,ShareEnc((v_159,SharedKey_160,v_161),ltkKASTGS[])
,ShareEnc((v_159,v_162),SharedKey_160)) ->
attacker:(v_159,ShareEnc((v_159,Kxaps[v_157,v_158,ShareEnc((v_159,Share
dKey_160,v_161),ltkKASTGS[]),ShareEnc((v_159,v_162),SharedKey_160),sid_
163],Tt[v_157,v_158,ShareEnc((v_159,SharedKey_160,v_161),ltkKASTGS[]),S
hareEnc((v_159,v_162),SharedKey_160),sid_163]),ltkAPSTGS[]),ShareEnc((v
_157,Kxaps[v_157,v_158,ShareEnc((v_159,SharedKey_160,v_161),ltkKASTGS[]
),ShareEnc((v_159,v_162),SharedKey_160),sid_163],v_158,Tt[v_157,v_158,S
```

```
hareEnc((v_159,SharedKey_160,v_161),ltkKASTGS[]),ShareEnc((v_159,v_162)
,SharedKey_160),sid_163]),SharedKey_160))
Rule 30:
attacker:(ShareEnc((v_177,SharedKey_178,v_179),ltkAPSTGS[]),ShareEnc((v
_177,v_180),SharedKey_178)) -> attacker:ShareEnc(v_180,SharedKey_178)
Rule 31:
attacker:(ShareEnc((v_182,SharedKey_183,v_184),ltkAPSTGS[]),ShareEnc((v
_182,v_185),SharedKey_183)) ->
end:sid_186,ProtocolAppServ(v_182,SharedKey_183)
Completing...
ok, secrecy assumption verified: fact unreachable attacker:ltkAKAS[]
ok, secrecy assumption verified: fact unreachable attacker:ltkKASTGS[]
ok, secrecy assumption verified: fact unreachable attacker:ltkAPSTGS[]
Starting query evinj:ProtocolAppServ(User_35,Kuaps_36) ==>
evinj:ProtocolTicServ2(User_35,Kuaps_36)
goal reachable: attacker:v_762 & attacker:v_763 ->
end:endsid_764,ProtocolAppServ(UserA[],Kxaps[AppServX[],Nat[(UserA[]),v
_763,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_765]),sid_766
],Nak[sid_765],Tk[(UserA[],TicServA[],Nak[sid_765]),sid_766]),ltkAKAS[]
),sid_765],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_765]),sid_
766],Tk[(UserA[],TicServA[],Nak[sid_765]),sid_766]),ltkKASTGS[]),ShareE
nc((UserA[],Tat[(UserA[]),v_762,ShareEnc((TicServA[],Kxt[(UserA[],TicSe
rvA[],Nak[sid_765]),sid_766],Nak[sid_765],Tk[(UserA[],TicServA[],Nak[si
d_765]),sid_766]),ltkAKAS[]),sid_765]),Kxt[(UserA[],TicServA[],Nak[sid_
765]),sid_766]),sid_767])
rule 31
end:endsid_968,ProtocolAppServ(UserA[],Kxaps[AppServX[],Nat[(UserA[]),v
_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943
],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]
),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_
943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareE
nc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicSe
rvA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[si
d_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_
931]),sid_943]),sid_957])
  2-tuple
attacker:(ShareEnc((UserA[],Kxaps[AppServX[],Nat[(UserA[]),v_900,ShareE
```

nc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_9
31],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],
ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(Us
erA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[]
,Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[s
id_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid
_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_9
43]),sid_957],Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kx
t[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],T
icServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[]
,Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],
Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_
935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]
,Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[])
,sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]),lt
kAPSTGS[]),ShareEnc((UserA[],Tab[v_905,ShareEnc((AppServX[],Kxaps[AppSe
rvX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],
Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]
),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA
[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]
),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA
[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(User
A[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[
],TicServA[],Nak[sid_931]),sid_943]),sid_957],Nat[(UserA[]),v_900,Share
Enc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_
931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]
,Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],Ti
cServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak
[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[]
,TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931])
,sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc(
(TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931]
,Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kx
t[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]),Kxt[(UserA[],Ti
cServA[],Nak[sid_931]),sid_943]),(UserA[]),v_900,ShareEnc((TicServA[],K
xt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],
TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxaps[AppServX[

],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[
sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),si
d_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],N
ak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),lt
kKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],K
xt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],
TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],Ti
cServA[],Nak[sid_931]),sid_943]),sid_957]))

1-th

attacker:ShareEnc((UserA[],Kxaps[AppServX[],Nat[(UserA[]),v_900,ShareEn
c((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_93
1],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],S
hareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(Use
rA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],
Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[si
d_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_
943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_94
3]),sid_957],Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt
[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],Ti
cServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],
Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],N
ak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_9
35,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],
Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),
sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]),ltk
APSTGS[])

duplicate

attacker:(UserA[],ShareEnc((UserA[],Kxaps[AppServX[],Nat[(UserA[]),v_90
0,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],N
ak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),s
id_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943
],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc(
(UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA
[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_9
31]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931
]),sid_943]),sid_957],Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicSe
rvA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(U

serA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc(
(UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],Tic
ServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(Use
rA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),
sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),lt
kAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_
957]),ltkAPSTGS[]),ShareEnc((AppServX[],Kxaps[AppServX[],Nat[(UserA[]),
v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_94
3],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[
]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid
_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),Share
Enc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicS
ervA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[s
id_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid
_931]),sid_943]),sid_957],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[
(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],Tic
ServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],Tt[AppServX[],Nat[(
UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931
]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943])
,ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_
931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS
[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(Use
rA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServ
A[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[
],Nak[sid_931]),sid_943]),sid_957]),Kxt[(UserA[],TicServA[],Nak[sid_931
]),sid_943]))

1-th

attacker:ShareEnc((UserA[],Tab[v_905,ShareEnc((AppServX[],Kxaps[AppServ
X[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Na
k[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),
sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[]
,Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),
ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[]
,Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[
],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],
TicServA[],Nak[sid_931]),sid_943]),sid_957],Nat[(UserA[]),v_900,ShareEn
c((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_93

1],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],T
t[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicS
ervA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[s
id_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],T
icServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),s
id_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((T
icServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],T
k[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[
(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]),Kxt[(UserA[],TicS
ervA[],Nak[sid_931]),sid_943]),(UserA[]),v_900,ShareEnc((TicServA[],Kxt
[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],Ti
cServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxaps[AppServX[],
Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[si
d_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_
943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak
[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkK
ASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt
[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],Ti
cServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicS
ervA[],Nak[sid_931]),sid_943]),sid_957])

rule 26

attacker:(v_905,ShareEnc((UserA[],Tab[v_905,ShareEnc((AppServX[],Kxaps[
AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicSer
vA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid
_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],Tic
ServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid
_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((Tic
ServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[
(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(U
serA[],TicServA[],Nak[sid_931]),sid_943]),sid_957],Nat[(UserA[]),v_900,
ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak
[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid
_931],Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA
[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[
],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(Us
erA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_
931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,Shar

eEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxaps[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]))

3-tuple

attacker:(UserA[],v_905,ShareEnc((AppServX[],Kxaps[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]))

duplicate attacker:UserA[]

any attacker:v_905

2-th

attacker:ShareEnc((AppServX[],Kxaps[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943])

rule 29

attacker:(UserA[],ShareEnc((UserA[],Kxaps[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_957],Tt[AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),lt

kAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]),sid_
957]),ltkAPSTGS[]),ShareEnc((AppServX[],Kxaps[AppServX[],Nat[(UserA[]),
v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_94
3],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[
]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid
_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),Share
Enc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicS
ervA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[s
id_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid
_931]),sid_943]),sid_957],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[
(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],Tic
ServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],Tt[AppServX[],Nat[(
UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931
]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943])
,ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_
931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS
[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(Use
rA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServ
A[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[
],Nak[sid_931]),sid_943]),sid_957]),Kxt[(UserA[],TicServA[],Nak[sid_931
]),sid_943]))

4-tuple

attacker:(AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(User
A[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA
[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],ShareEnc((UserA[],Kxt[(U
serA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid
_931]),sid_943]),ltkKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_935,Sha
reEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[si
d_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_93
1]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]))

rule 20 attacker:AppServX[]

1-th

attacker:Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA
[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_9
31]),sid_943]),ltkAKAS[]),sid_931]

rule 25

attacker:(AppServX[],Nat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(User

A[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],v_900,ShareEnc((UserA[],Tat[(UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]))

duplicate

attacker:((UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]))

1-th

attacker:ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[])

duplicate

attacker:(UserA[],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[]),ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]))

3-th

attacker:ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943])

rule 25

attacker:(AppServX[],Nat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931],v_935,ShareEnc((UserA[],Tat[(UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]),sid_931]),Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943]))

3-tuple

attacker:((UserA[]),v_935,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]))

duplicate attacker:(UserA[])

hypothesis attacker:v_935

```
                    duplicate
attacker:ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid
_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAK
AS[])
        3-tuple
attacker:((UserA[]),v_900,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],
Nak[sid_931]),sid_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]
),sid_943]),ltkAKAS[]))
          1-tuple attacker:(UserA[])
            duplicate attacker:UserA[]
          hypothesis attacker:v_900
          2-th
attacker:ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid
_943],Nak[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAK
AS[])
            rule 28
attacker:(UserA[],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_931
]),sid_943],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkKASTGS[])
,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_931]),sid_943],Na
k[sid_931],Tk[(UserA[],TicServA[],Nak[sid_931]),sid_943]),ltkAKAS[]))
              3-tuple attacker:(UserA[],TicServA[],Nak[sid_931])
                rule 21 attacker:UserA[]
                rule 19 attacker:TicServA[]
                2-th attacker:Nak[sid_931]
                  rule 24 attacker:(UserA[],TicServA[],Nak[sid_931])

Could not find an attack corresponding to this derivation.
RESULT evinj:ProtocolAppServ(User_35,Kuaps_36) ==>
evinj:ProtocolTicServ2(User_35,Kuaps_36) cannot be proved.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_1081,v_1081
Rule 1: attacker:v_1084 & attacker:v_1083 ->
attacker:ShareEnc(v_1084,v_1083)
Rule 2: attacker:ShareEnc(Msg_1085,SharedKey_1086) &
attacker:SharedKey_1086 -> attacker:Msg_1085
```

```
Rule 3: attacker:v_1089 & attacker:v_1088 & attacker:v_1087 ->
attacker:(v_1089,v_1088,v_1087)
Rule 4: attacker:(v_1092,v_1091,v_1090) -> attacker:v_1092
Rule 5: attacker:(v_1095,v_1094,v_1093) -> attacker:v_1094
Rule 6: attacker:(v_1098,v_1097,v_1096) -> attacker:v_1096
Rule 7: attacker:v_1099 -> attacker:(v_1099)
Rule 8: attacker:(v_1100) -> attacker:v_1100
Rule 9: attacker:v_1104 & attacker:v_1103 & attacker:v_1102 &
attacker:v_1101 -> attacker:(v_1104,v_1103,v_1102,v_1101)
Rule 10: attacker:(v_1108,v_1107,v_1106,v_1105) -> attacker:v_1108
Rule 11: attacker:(v_1112,v_1111,v_1110,v_1109) -> attacker:v_1111
Rule 12: attacker:(v_1116,v_1115,v_1114,v_1113) -> attacker:v_1114
Rule 13: attacker:(v_1120,v_1119,v_1118,v_1117) -> attacker:v_1117
Rule 14: attacker:v_1122 & attacker:v_1121 -> attacker:(v_1122,v_1121)
Rule 15: attacker:(v_1124,v_1123) -> attacker:v_1124
Rule 16: attacker:(v_1126,v_1125) -> attacker:v_1125
Rule 17: mess:v_1128,v_1127 & attacker:v_1128 -> attacker:v_1127
Rule 18: attacker:v_1130 & attacker:v_1129 -> mess:v_1130,v_1129
Rule 19: attacker:TicServA[]
Rule 20: attacker:AppServX[]
Rule 21: attacker:UserA[]
Rule 22: attacker:chnl[]
Rule 23: attacker:new_name[v_1131]
Rule 24: attacker:(UserA[],TicServA[],Nak[sid_1133])
Rule 25:
attacker:((UserA[]),v_1145,ShareEnc((TicServA[],v_1146,Nak[sid_1147],v_
1148),ltkAKAS[])) ->
attacker:(AppServX[],Nat[(UserA[]),v_1145,ShareEnc((TicServA[],v_1146,N
ak[sid_1147],v_1148),ltkAKAS[]),sid_1147],v_1145,ShareEnc((UserA[],Tat[
(UserA[]),v_1145,ShareEnc((TicServA[],v_1146,Nak[sid_1147],v_1148),ltkA
KAS[]),sid_1147]),v_1146))
Rule 26:
attacker:(UserA[],v_1159,ShareEnc((AppServX[],v_1160,Nat[(UserA[]),v_11
61,ShareEnc((TicServA[],SharedKey_1162,Nak[sid_1163],v_1164),ltkAKAS[])
,sid_1163],v_1165),SharedKey_1162)) &
attacker:((UserA[]),v_1161,ShareEnc((TicServA[],SharedKey_1162,Nak[sid_
1163],v_1164),ltkAKAS[])) ->
```

```
attacker:(v_1159,ShareEnc((UserA[],Tab[v_1159,ShareEnc((AppServX[],v_11
60,Nat[(UserA[]),v_1161,ShareEnc((TicServA[],SharedKey_1162,Nak[sid_116
3],v_1164),ltkAKAS[]),sid_1163],v_1165),SharedKey_1162),(UserA[]),v_116
1,ShareEnc((TicServA[],SharedKey_1162,Nak[sid_1163],v_1164),ltkAKAS[]),
sid_1163]),v_1160))
Rule 27:
attacker:ShareEnc((Tab[v_1170,ShareEnc((AppServX[],SharedKey_1171,Nat[(
UserA[]),v_1172,ShareEnc((TicServA[],SharedKey_1173,Nak[sid_1174],v_117
5),ltkAKAS[]),sid_1174],v_1176),SharedKey_1173),(UserA[]),v_1172,ShareE
nc((TicServA[],SharedKey_1173,Nak[sid_1174],v_1175),ltkAKAS[]),sid_1174
]),SharedKey_1171) &
attacker:(UserA[],v_1170,ShareEnc((AppServX[],SharedKey_1171,Nat[(UserA
[]),v_1172,ShareEnc((TicServA[],SharedKey_1173,Nak[sid_1174],v_1175),lt
kAKAS[]),sid_1174],v_1176),SharedKey_1173)) &
attacker:((UserA[]),v_1172,ShareEnc((TicServA[],SharedKey_1173,Nak[sid_
1174],v_1175),ltkAKAS[])) ->
attacker:ShareEnc(AttackSuccess[],SharedKey_1171)
Rule 28: attacker:(v_1182,v_1183,v_1184) ->
attacker:(v_1182,ShareEnc((v_1182,Kxt[(v_1182,v_1183,v_1184),sid_1185],
Tk[(v_1182,v_1183,v_1184),sid_1185]),ltkKASTGS[]),ShareEnc((v_1183,Kxt[
(v_1182,v_1183,v_1184),sid_1185],v_1184,Tk[(v_1182,v_1183,v_1184),sid_1
185]),ltkAKAS[]))
Rule 29:
attacker:(v_1201,v_1202,ShareEnc((v_1203,SharedKey_1204,v_1205),ltkKAST
GS[]),ShareEnc((v_1203,v_1206),SharedKey_1204)) ->
end:sid_1207,ProtocolTicServ1(v_1203,SharedKey_1204)
Rule 30:
attacker:(v_1208,v_1209,ShareEnc((v_1210,SharedKey_1211,v_1212),ltkKAST
GS[]),ShareEnc((v_1210,v_1213),SharedKey_1211)) ->
attacker:(v_1210,ShareEnc((v_1210,Kxaps[v_1208,v_1209,ShareEnc((v_1210,
SharedKey_1211,v_1212),ltkKASTGS[]),ShareEnc((v_1210,v_1213),SharedKey_
1211),sid_1214],Tt[v_1208,v_1209,ShareEnc((v_1210,SharedKey_1211,v_1212
),ltkKASTGS[]),ShareEnc((v_1210,v_1213),SharedKey_1211),sid_1214]),ltkA
PSTGS[]),ShareEnc((v_1208,Kxaps[v_1208,v_1209,ShareEnc((v_1210,SharedKe
y_1211,v_1212),ltkKASTGS[]),ShareEnc((v_1210,v_1213),SharedKey_1211),si
d_1214],v_1209,Tt[v_1208,v_1209,ShareEnc((v_1210,SharedKey_1211,v_1212)
```

```
,ltkKASTGS[]),ShareEnc((v_1210,v_1213),SharedKey_1211),sid_1214]),Share
dKey_1211))
Rule 31:
attacker:(ShareEnc((v_1228,SharedKey_1229,v_1230),ltkAPSTGS[]),ShareEnc
((v_1228,v_1231),SharedKey_1229)) ->
attacker:ShareEnc(v_1231,SharedKey_1229)
Completing...
ok, secrecy assumption verified: fact unreachable attacker:ltkAKAS[]
ok, secrecy assumption verified: fact unreachable attacker:ltkKASTGS[]
ok, secrecy assumption verified: fact unreachable attacker:ltkAPSTGS[]
Starting query evinj:ProtocolTicServ1(User_1079,Kutgs_1080) ==>
evinj:ProtocolAuthServ(User_1079,Kutgs_1080)
goal reachable:
end:endsid_1753,ProtocolTicServ1(UserA[],Kxt[(UserA[],TicServA[],Nak[si
d_1754]),sid_1755])
rule 29
end:endsid_1810,ProtocolTicServ1(UserA[],Kxt[(UserA[],TicServA[],Nak[si
d_1783]),sid_1795])
  4-tuple
attacker:(v_1803,v_1804,ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[s
id_1783]),sid_1795],Tk[(UserA[],TicServA[],Nak[sid_1783]),sid_1795]),lt
kKASTGS[]),ShareEnc((UserA[],Tat[(UserA[]),v_1787,ShareEnc((TicServA[],
Kxt[(UserA[],TicServA[],Nak[sid_1783]),sid_1795],Nak[sid_1783],Tk[(User
A[],TicServA[],Nak[sid_1783]),sid_1795]),ltkAKAS[]),sid_1783]),Kxt[(Use
rA[],TicServA[],Nak[sid_1783]),sid_1795]))
    any attacker:v_1803
    any attacker:v_1804
    1-th
attacker:ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_1783]),sid_1
795],Tk[(UserA[],TicServA[],Nak[sid_1783]),sid_1795]),ltkKASTGS[])
      duplicate
attacker:(UserA[],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_178
3]),sid_1795],Tk[(UserA[],TicServA[],Nak[sid_1783]),sid_1795]),ltkKASTG
S[]),ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_1783]),sid_17
95],Nak[sid_1783],Tk[(UserA[],TicServA[],Nak[sid_1783]),sid_1795]),ltkA
KAS[]))
```

```
    3-th
attacker:ShareEnc((UserA[],Tat[(UserA[]),v_1787,ShareEnc((TicServA[],Kx
t[(UserA[],TicServA[],Nak[sid_1783]),sid_1795],Nak[sid_1783],Tk[(UserA[
],TicServA[],Nak[sid_1783]),sid_1795]),ltkAKAS[]),sid_1783]),Kxt[(UserA
[],TicServA[],Nak[sid_1783]),sid_1795])
      rule 25
attacker:(AppServX[],Nat[(UserA[]),v_1787,ShareEnc((TicServA[],Kxt[(Use
rA[],TicServA[],Nak[sid_1783]),sid_1795],Nak[sid_1783],Tk[(UserA[],TicS
ervA[],Nak[sid_1783]),sid_1795]),ltkAKAS[]),sid_1783],v_1787,ShareEnc((
UserA[],Tat[(UserA[]),v_1787,ShareEnc((TicServA[],Kxt[(UserA[],TicServA
[],Nak[sid_1783]),sid_1795],Nak[sid_1783],Tk[(UserA[],TicServA[],Nak[si
d_1783]),sid_1795]),ltkAKAS[]),sid_1783]),Kxt[(UserA[],TicServA[],Nak[s
id_1783]),sid_1795]))
        3-tuple
attacker:((UserA[]),v_1787,ShareEnc((TicServA[],Kxt[(UserA[],TicServA[]
,Nak[sid_1783]),sid_1795],Nak[sid_1783],Tk[(UserA[],TicServA[],Nak[sid_
1783]),sid_1795]),ltkAKAS[]))
          1-tuple attacker:(UserA[])
            duplicate attacker:UserA[]
          any attacker:v_1787
          2-th
attacker:ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_1783]),si
d_1795],Nak[sid_1783],Tk[(UserA[],TicServA[],Nak[sid_1783]),sid_1795]),
ltkAKAS[])
            rule 28
attacker:(UserA[],ShareEnc((UserA[],Kxt[(UserA[],TicServA[],Nak[sid_178
3]),sid_1795],Tk[(UserA[],TicServA[],Nak[sid_1783]),sid_1795]),ltkKASTG
S[]),ShareEnc((TicServA[],Kxt[(UserA[],TicServA[],Nak[sid_1783]),sid_17
95],Nak[sid_1783],Tk[(UserA[],TicServA[],Nak[sid_1783]),sid_1795]),ltkA
KAS[]))
              3-tuple attacker:(UserA[],TicServA[],Nak[sid_1783])
                rule 21 attacker:UserA[]
                rule 19 attacker:TicServA[]
                2-th attacker:Nak[sid_1783]
                  rule 24 attacker:(UserA[],TicServA[],Nak[sid_1783])
```

Goal of the attack :

```
end:a_21[],ProtocolTicServ1(UserA[],Kxt_28[])
Initial state

Additional knowledge of the attacker:
      chnl
      UserA
      AppServX
      TicServA
      a_24
      a_25
      a_26

--------------------------------------------------------------
New processes:
      new ltkAKAS;
      new ltkKASTGS;
      new ltkAPSTGS;
      (
          !
          new Nak;
          out(chnl, (UserA,TicServA,Nak));
          in(chnl, (UserX_25,AuthServTicServ_26,AuthServUser_27));
          let (=UserA) = UserX_25 in
          let (=TicServA,Kat_28,=Nak,Tk_29) =
ShareDec(AuthServUser_27,ltkAKAS) in
          new Nat;
          new Tat;
          out(chnl,
(AppServX,Nat,AuthServTicServ_26,ShareEnc((UserA,Tat),Kat_28)));
          in(chnl, (=UserA,TicServAppServ_30,TicServUser_31));
          let (=AppServX,Kab_32,=Nat,Tt_33) =
ShareDec(TicServUser_31,Kat_28) in
          new Tab;
          out(chnl, (TicServAppServ_30,ShareEnc((UserA,Tab),Kab_32)));
          in(chnl, AppServUser_34);
          let (=Tab) = ShareDec(AppServUser_34,Kab_32) in
          event ProtocolCompleteUser(UserA,Kab_32);
```

```
          out(chnl, ShareEnc(AttackSuccess,Kab_32));
          0.
      ) | (
          (
              !
              in(chnl, UserAuthServ_21);
              let (UserX_22,TicServX_23,Nxk_24) = UserAuthServ_21 in
              new Kxt;
              new Tk;
              out(chnl,
(UserX_22,ShareEnc((UserX_22,Kxt,Tk),ltkKASTGS),ShareEnc((TicServX_23,K
xt,Nxk_24,Tk),ltkAKAS)));
              event ProtocolAuthServ(UserX_22,Kxt);
              0.
          ) | (
              (
                  !
                  in(chnl,
(AppServX_13,Nxt_14,AuthServTicServ_15,UserTicServ_16));
                  let (UserX_17,Kxt_18,Tk_19) =
ShareDec(AuthServTicServ_15,ltkKASTGS) in
                  let (=UserX_17,Txt_20) =
ShareDec(UserTicServ_16,Kxt_18) in
                  event ProtocolTicServ1(UserX_17,Kxt_18);
                  new Kxaps;
                  new Tt;
                  out(chnl,
(UserX_17,ShareEnc((UserX_17,Kxaps,Tt),ltkAPSTGS),ShareEnc((AppServX_13
,Kxaps,Nxt_14,Tt),Kxt_18)));
                  event ProtocolTicServ2(UserX_17,Kxaps);
                  0.
              ) | (
                  !
                  in(chnl, (TicServAppServ_7,UserAppServ_8));
                  let (UserX_9,Kxaps_10,Tx_11) =
ShareDec(TicServAppServ_7,ltkAPSTGS) in
```

```
                    let (=UserX_9,Txaps_12) =
ShareDec(UserAppServ_8,Kxaps_10) in
                    out(chnl, ShareEnc(Txaps_12,Kxaps_10));
                    event ProtocolAppServ(UserX_9,Kxaps_10);
                    0.
                )
           )
      )

------------------------------------------------------------
1st process: New ltkAKAS creating ltkAKAS_31

1st process: New ltkKASTGS creating ltkKASTGS_30

1st process: New ltkAPSTGS creating ltkAPSTGS_34

1st process: Reduction |

2nd process: Reduction |

3rd process: Reduction |

4th process: Reduction ! 0 copy(ies)

3rd process: Reduction ! 1 copy(ies)

2nd process: Reduction ! 1 copy(ies)

1st process: Reduction ! 1 copy(ies)

1st process: New Nak creating Nak_27

1st process: Out(chnl, (UserA,TicServA,Nak_27)) done

Additional knowledge of the attacker:
      Nak_27
```

```
------------------------------------------------------------
New processes:
      in(chnl, (UserX_1829,AuthServTicServ_1830,AuthServUser_1831));
      let (=UserA) = UserX_1829 in
      let (=TicServA,Kat_1832,=Nak_27,Tk_1833) =
ShareDec(AuthServUser_1831,ltkAKAS_31) in
      new Nat;
      new Tat;
      out(chnl,
(AppServX,Nat,AuthServTicServ_1830,ShareEnc((UserA,Tat),Kat_1832)));
      in(chnl, (=UserA,TicServAppServ_1834,TicServUser_1835));
      let (=AppServX,Kab_1836,=Nat,Tt_1837) =
ShareDec(TicServUser_1835,Kat_1832) in
      new Tab;
      out(chnl, (TicServAppServ_1834,ShareEnc((UserA,Tab),Kab_1836)));
      in(chnl, AppServUser_1838);
      let (=Tab) = ShareDec(AppServUser_1838,Kab_1836) in
      event ProtocolCompleteUser(UserA,Kab_1836);
      out(chnl, ShareEnc(AttackSuccess,Kab_1836));
      0.

|
      in(chnl, UserAuthServ_1824);
      let (UserX_1825,TicServX_1826,Nxk_1827) = UserAuthServ_1824 in
      new Kxt;
      new Tk;
      out(chnl,
(UserX_1825,ShareEnc((UserX_1825,Kxt,Tk),ltkKASTGS_30),ShareEnc((TicSer
vX_1826,Kxt,Nxk_1827,Tk),ltkAKAS_31)));
      event ProtocolAuthServ(UserX_1825,Kxt);
      0.

|
      in(chnl,
(AppServX_1815,Nxt_1816,AuthServTicServ_1817,UserTicServ_1818));
      let (UserX_1819,Kxt_1820,Tk_1821) =
ShareDec(AuthServTicServ_1817,ltkKASTGS_30) in
```

```
      let (=UserX_1819,Txt_1822) = ShareDec(UserTicServ_1818,Kxt_1820)
in
      event ProtocolTicServ1(UserX_1819,Kxt_1820);
      new Kxaps;
      new Tt;
      out(chnl,
(UserX_1819,ShareEnc((UserX_1819,Kxaps,Tt),ltkAPSTGS_34),ShareEnc((AppS
ervX_1815,Kxaps,Nxt_1816,Tt),Kxt_1820)));
      event ProtocolTicServ2(UserX_1819,Kxaps);
      0.

------------------------------------------------------------
2nd process: In(chnl, UserAuthServ_1824) done with message
(UserA,TicServA,Nak_27)

2nd process: Let (UserX_1841,TicServX_1842,Nxk_1843) =
(UserA,TicServA,Nak_27) succeeds

2nd process: New Kxt creating Kxt_28

2nd process: New Tk creating Tk_29

2nd process: Out(chnl,
(UserA,ShareEnc((UserA,Kxt_28,Tk_29),ltkKASTGS_30),ShareEnc((TicServA,K
xt_28,Nak_27,Tk_29),ltkAKAS_31))) done

Additional knowledge of the attacker:
      ShareEnc((TicServA,Kxt_28,Nak_27,Tk_29),ltkAKAS_31)
      ShareEnc((UserA,Kxt_28,Tk_29),ltkKASTGS_30)

------------------------------------------------------------
2nd process: Event(ProtocolAuthServ(UserA,Kxt_28)) destructor fails or
event blocks

New processes:
      in(chnl, (UserX_1829,AuthServTicServ_1830,AuthServUser_1831));
      let (=UserA) = UserX_1829 in
```

```
      let (=TicServA,Kat_1832,=Nak_27,Tk_1833) =
ShareDec(AuthServUser_1831,ltkAKAS_31) in
      new Nat;
      new Tat;
      out(chnl,
(AppServX,Nat,AuthServTicServ_1830,ShareEnc((UserA,Tat),Kat_1832)));
      in(chnl, (=UserA,TicServAppServ_1834,TicServUser_1835));
      let (=AppServX,Kab_1836,=Nat,Tt_1837) =
ShareDec(TicServUser_1835,Kat_1832) in
      new Tab;
      out(chnl, (TicServAppServ_1834,ShareEnc((UserA,Tab),Kab_1836)));
      in(chnl, AppServUser_1838);
      let (=Tab) = ShareDec(AppServUser_1838,Kab_1836) in
      event ProtocolCompleteUser(UserA,Kab_1836);
      out(chnl, ShareEnc(AttackSuccess,Kab_1836));
      0.

|
      in(chnl,
(AppServX_1815,Nxt_1816,AuthServTicServ_1817,UserTicServ_1818));
      let (UserX_1819,Kxt_1820,Tk_1821) =
ShareDec(AuthServTicServ_1817,ltkKASTGS_30) in
      let (=UserX_1819,Txt_1822) = ShareDec(UserTicServ_1818,Kxt_1820)
in
      event ProtocolTicServ1(UserX_1819,Kxt_1820);
      new Kxaps;
      new Tt;
      out(chnl,
(UserX_1819,ShareEnc((UserX_1819,Kxaps,Tt),ltkAPSTGS_34),ShareEnc((AppS
ervX_1815,Kxaps,Nxt_1816,Tt),Kxt_1820)));
      event ProtocolTicServ2(UserX_1819,Kxaps);
      0.

------------------------------------------------------------
1st process: In(chnl,
(UserX_1829,AuthServTicServ_1830,AuthServUser_1831)) done with message
((UserA),a_26,ShareEnc((TicServA,Kxt_28,Nak_27,Tk_29),ltkAKAS_31))
```

```
1st process: Let (=UserA) = (UserA) succeeds

1st process: Let (=TicServA,Kat_1858,=Nak_27,Tk_1859) =
ShareDec(ShareEnc((TicServA,Kxt_28,Nak_27,Tk_29),ltkAKAS_31),ltkAKAS_31
) succeeds

1st process: New Nat creating Nat_32

1st process: New Tat creating Tat_33

1st process: Out(chnl,
(AppServX,Nat_32,a_26,ShareEnc((UserA,Tat_33),Kxt_28))) done

Additional knowledge of the attacker:
      ShareEnc((UserA,Tat_33),Kxt_28)
      Nat_32

------------------------------------------------------------
New processes:
      in(chnl, (=UserA,TicServAppServ_1871,TicServUser_1872));
      let (=AppServX,Kab_1873,=Nat_32,Tt_1874) =
ShareDec(TicServUser_1872,Kxt_28) in
      new Tab;
      out(chnl, (TicServAppServ_1871,ShareEnc((UserA,Tab),Kab_1873)));
      in(chnl, AppServUser_1875);
      let (=Tab) = ShareDec(AppServUser_1875,Kab_1873) in
      event ProtocolCompleteUser(UserA,Kab_1873);
      out(chnl, ShareEnc(AttackSuccess,Kab_1873));
      0.

|
      in(chnl,
(AppServX_1815,Nxt_1816,AuthServTicServ_1817,UserTicServ_1818));
      let (UserX_1819,Kxt_1820,Tk_1821) =
ShareDec(AuthServTicServ_1817,ltkKASTGS_30) in
```

```
      let (=UserX_1819,Txt_1822) = ShareDec(UserTicServ_1818,Kxt_1820)
in
      event ProtocolTicServ1(UserX_1819,Kxt_1820);
      new Kxaps;
      new Tt;
      out(chnl,
(UserX_1819,ShareEnc((UserX_1819,Kxaps,Tt),ltkAPSTGS_34),ShareEnc((AppS
ervX_1815,Kxaps,Nxt_1816,Tt),Kxt_1820)));
      event ProtocolTicServ2(UserX_1819,Kxaps);
      0.

------------------------------------------------------------
2nd process: In(chnl,
(AppServX_1815,Nxt_1816,AuthServTicServ_1817,UserTicServ_1818)) done
with message
(a_24,a_25,ShareEnc((UserA,Kxt_28,Tk_29),ltkKASTGS_30),ShareEnc((UserA,
Tat_33),Kxt_28))

2nd process: Let (UserX_1882,Kxt_1883,Tk_1884) =
ShareDec(ShareEnc((UserA,Kxt_28,Tk_29),ltkKASTGS_30),ltkKASTGS_30)
succeeds

2nd process: Let (=UserA,Txt_1891) =
ShareDec(ShareEnc((UserA,Tat_33),Kxt_28),Kxt_28) succeeds

2nd process: Event(ProtocolTicServ1(UserA,Kxt_28)) is the goal

New processes:
      in(chnl, (=UserA,TicServAppServ_1871,TicServUser_1872));
      let (=AppServX,Kab_1873,=Nat_32,Tt_1874) =
ShareDec(TicServUser_1872,Kxt_28) in
      new Tab;
      out(chnl, (TicServAppServ_1871,ShareEnc((UserA,Tab),Kab_1873)));
      in(chnl, AppServUser_1875);
      let (=Tab) = ShareDec(AppServUser_1875,Kab_1873) in
      event ProtocolCompleteUser(UserA,Kab_1873);
      out(chnl, ShareEnc(AttackSuccess,Kab_1873));
```

```
      0.

|
      new Kxaps;
      new Tt;
      out(chnl,
(UserA,ShareEnc((UserA,Kxaps,Tt),ltkAPSTGS_34),ShareEnc((a_24,Kxaps,a_2
5,Tt),Kxt_28)));
      event ProtocolTicServ2(UserA,Kxaps);
      0.

------------------------------------------------------------
An attack has been found.
RESULT evinj:ProtocolTicServ1(User_1079,Kutgs_1080) ==>
evinj:ProtocolAuthServ(User_1079,Kutgs_1080) is false.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_1899,v_1899
Rule 1: attacker:v_1902 & attacker:v_1901 ->
attacker:ShareEnc(v_1902,v_1901)
Rule 2: attacker:ShareEnc(Msg_1903,SharedKey_1904) &
attacker:SharedKey_1904 -> attacker:Msg_1903
Rule 3: attacker:v_1907 & attacker:v_1906 & attacker:v_1905 ->
attacker:(v_1907,v_1906,v_1905)
Rule 4: attacker:(v_1910,v_1909,v_1908) -> attacker:v_1910
Rule 5: attacker:(v_1913,v_1912,v_1911) -> attacker:v_1912
Rule 6: attacker:(v_1916,v_1915,v_1914) -> attacker:v_1914
Rule 7: attacker:v_1917 -> attacker:(v_1917)
Rule 8: attacker:(v_1918) -> attacker:v_1918
Rule 9: attacker:v_1922 & attacker:v_1921 & attacker:v_1920 &
attacker:v_1919 -> attacker:(v_1922,v_1921,v_1920,v_1919)
Rule 10: attacker:(v_1926,v_1925,v_1924,v_1923) -> attacker:v_1926
Rule 11: attacker:(v_1930,v_1929,v_1928,v_1927) -> attacker:v_1929
Rule 12: attacker:(v_1934,v_1933,v_1932,v_1931) -> attacker:v_1932
Rule 13: attacker:(v_1938,v_1937,v_1936,v_1935) -> attacker:v_1935
Rule 14: attacker:v_1940 & attacker:v_1939 -> attacker:(v_1940,v_1939)
Rule 15: attacker:(v_1942,v_1941) -> attacker:v_1942
```

```
Rule 16: attacker:(v_1944,v_1943) -> attacker:v_1943
Rule 17: mess:v_1946,v_1945 & attacker:v_1946 -> attacker:v_1945
Rule 18: attacker:v_1948 & attacker:v_1947 -> mess:v_1948,v_1947
Rule 19: attacker:TicServA[]
Rule 20: attacker:AppServX[]
Rule 21: attacker:UserA[]
Rule 22: attacker:chnl[]
Rule 23: attacker:new_name[v_1949]
Rule 24: attacker:(UserA[],TicServA[],Nak[sid_1951])
Rule 25:
attacker:((UserA[]),v_1963,ShareEnc((TicServA[],v_1964,Nak[sid_1965],v_
1966),ltkAKAS[])) ->
attacker:(AppServX[],Nat[(UserA[]),v_1963,ShareEnc((TicServA[],v_1964,N
ak[sid_1965],v_1966),ltkAKAS[]),sid_1965],v_1963,ShareEnc((UserA[],Tat[
(UserA[]),v_1963,ShareEnc((TicServA[],v_1964,Nak[sid_1965],v_1966),ltkA
KAS[]),sid_1965]),v_1964))
Rule 26:
attacker:(UserA[],v_1977,ShareEnc((AppServX[],v_1978,Nat[(UserA[]),v_19
79,ShareEnc((TicServA[],SharedKey_1980,Nak[sid_1981],v_1982),ltkAKAS[])
,sid_1981],v_1983),SharedKey_1980)) &
attacker:((UserA[]),v_1979,ShareEnc((TicServA[],SharedKey_1980,Nak[sid_
1981],v_1982),ltkAKAS[])) ->
attacker:(v_1977,ShareEnc((UserA[],Tab[v_1977,ShareEnc((AppServX[],v_19
78,Nat[(UserA[]),v_1979,ShareEnc((TicServA[],SharedKey_1980,Nak[sid_198
1],v_1982),ltkAKAS[]),sid_1981],v_1983),SharedKey_1980),(UserA[]),v_197
9,ShareEnc((TicServA[],SharedKey_1980,Nak[sid_1981],v_1982),ltkAKAS[]),
sid_1981]),v_1978))
Rule 27:
attacker:ShareEnc((Tab[v_1988,ShareEnc((AppServX[],SharedKey_1989,Nat[(
UserA[]),v_1990,ShareEnc((TicServA[],SharedKey_1991,Nak[sid_1992],v_199
3),ltkAKAS[]),sid_1992],v_1994),SharedKey_1991),(UserA[]),v_1990,ShareE
nc((TicServA[],SharedKey_1991,Nak[sid_1992],v_1993),ltkAKAS[]),sid_1992
]),SharedKey_1989) &
attacker:(UserA[],v_1988,ShareEnc((AppServX[],SharedKey_1989,Nat[(UserA
[]),v_1990,ShareEnc((TicServA[],SharedKey_1991,Nak[sid_1992],v_1993),lt
kAKAS[]),sid_1992],v_1994),SharedKey_1991)) &
attacker:((UserA[]),v_1990,ShareEnc((TicServA[],SharedKey_1991,Nak[sid_
```

```
1992],v_1993),ltkAKAS[])) ->
end:sid_1992,ProtocolCompleteUser(UserA[],SharedKey_1989)
Rule 28:
attacker:ShareEnc((Tab[v_1995,ShareEnc((AppServX[],SharedKey_1996,Nat[(
UserA[]),v_1997,ShareEnc((TicServA[],SharedKey_1998,Nak[sid_1999],v_200
0),ltkAKAS[]),sid_1999],v_2001),SharedKey_1998),(UserA[]),v_1997,ShareE
nc((TicServA[],SharedKey_1998,Nak[sid_1999],v_2000),ltkAKAS[]),sid_1999
]),SharedKey_1996) &
attacker:(UserA[],v_1995,ShareEnc((AppServX[],SharedKey_1996,Nat[(UserA
[]),v_1997,ShareEnc((TicServA[],SharedKey_1998,Nak[sid_1999],v_2000),lt
kAKAS[]),sid_1999],v_2001),SharedKey_1998)) &
attacker:((UserA[]),v_1997,ShareEnc((TicServA[],SharedKey_1998,Nak[sid_
1999],v_2000),ltkAKAS[])) ->
attacker:ShareEnc(AttackSuccess[],SharedKey_1996)
Rule 29: attacker:(v_2007,v_2008,v_2009) ->
attacker:(v_2007,ShareEnc((v_2007,Kxt[(v_2007,v_2008,v_2009),sid_2010],
Tk[(v_2007,v_2008,v_2009),sid_2010]),ltkKASTGS[]),ShareEnc((v_2008,Kxt[
(v_2007,v_2008,v_2009),sid_2010],v_2009,Tk[(v_2007,v_2008,v_2009),sid_2
010]),ltkAKAS[]))
Rule 30:
attacker:(v_2026,v_2027,ShareEnc((v_2028,SharedKey_2029,v_2030),ltkKAST
GS[]),ShareEnc((v_2028,v_2031),SharedKey_2029)) ->
attacker:(v_2028,ShareEnc((v_2028,Kxaps[v_2026,v_2027,ShareEnc((v_2028,
SharedKey_2029,v_2030),ltkKASTGS[]),ShareEnc((v_2028,v_2031),SharedKey_
2029),sid_2032],Tt[v_2026,v_2027,ShareEnc((v_2028,SharedKey_2029,v_2030
),ltkKASTGS[]),ShareEnc((v_2028,v_2031),SharedKey_2029),sid_2032]),ltkA
PSTGS[]),ShareEnc((v_2026,Kxaps[v_2026,v_2027,ShareEnc((v_2028,SharedKe
y_2029,v_2030),ltkKASTGS[]),ShareEnc((v_2028,v_2031),SharedKey_2029),si
d_2032],v_2027,Tt[v_2026,v_2027,ShareEnc((v_2028,SharedKey_2029,v_2030)
,ltkKASTGS[]),ShareEnc((v_2028,v_2031),SharedKey_2029),sid_2032]),Share
dKey_2029))
Rule 31:
attacker:(ShareEnc((v_2046,SharedKey_2047,v_2048),ltkAPSTGS[]),ShareEnc
((v_2046,v_2049),SharedKey_2047)) ->
attacker:ShareEnc(v_2049,SharedKey_2047)
Completing...
ok, secrecy assumption verified: fact unreachable attacker:ltkAKAS[]
```

```
ok, secrecy assumption verified: fact unreachable attacker:ltkKASTGS[]
ok, secrecy assumption verified: fact unreachable attacker:ltkAPSTGS[]
Starting query evinj:ProtocolCompleteUser(User_1896,Kuaps_1897) ==>
(evinj:ProtocolTicServ2(User_1896,Kuaps_1897) &
evinj:ProtocolAuthServ(User_1896,Katgs_1898))
RESULT evinj:ProtocolCompleteUser(User_1896,Kuaps_1897) ==>
(evinj:ProtocolTicServ2(User_1896,Kuaps_1897) &
evinj:ProtocolAuthServ(User_1896,Katgs_1898)) is true.
-- Secrecy & events.
Starting rules:
Rule 0: equal:v_2551,v_2551
Rule 1: attacker:v_2554 & attacker:v_2553 ->
attacker:ShareEnc(v_2554,v_2553)
Rule 2: attacker:ShareEnc(Msg_2555,SharedKey_2556) &
attacker:SharedKey_2556 -> attacker:Msg_2555
Rule 3: attacker:v_2559 & attacker:v_2558 & attacker:v_2557 ->
attacker:(v_2559,v_2558,v_2557)
Rule 4: attacker:(v_2562,v_2561,v_2560) -> attacker:v_2562
Rule 5: attacker:(v_2565,v_2564,v_2563) -> attacker:v_2564
Rule 6: attacker:(v_2568,v_2567,v_2566) -> attacker:v_2566
Rule 7: attacker:v_2569 -> attacker:(v_2569)
Rule 8: attacker:(v_2570) -> attacker:v_2570
Rule 9: attacker:v_2574 & attacker:v_2573 & attacker:v_2572 &
attacker:v_2571 -> attacker:(v_2574,v_2573,v_2572,v_2571)
Rule 10: attacker:(v_2578,v_2577,v_2576,v_2575) -> attacker:v_2578
Rule 11: attacker:(v_2582,v_2581,v_2580,v_2579) -> attacker:v_2581
Rule 12: attacker:(v_2586,v_2585,v_2584,v_2583) -> attacker:v_2584
Rule 13: attacker:(v_2590,v_2589,v_2588,v_2587) -> attacker:v_2587
Rule 14: attacker:v_2592 & attacker:v_2591 -> attacker:(v_2592,v_2591)
Rule 15: attacker:(v_2594,v_2593) -> attacker:v_2594
Rule 16: attacker:(v_2596,v_2595) -> attacker:v_2595
Rule 17: mess:v_2598,v_2597 & attacker:v_2598 -> attacker:v_2597
Rule 18: attacker:v_2600 & attacker:v_2599 -> mess:v_2600,v_2599
Rule 19: attacker:TicServA[]
Rule 20: attacker:AppServX[]
Rule 21: attacker:UserA[]
Rule 22: attacker:chnl[]
```

```
Rule 23: attacker:new_name[v_2601]
Rule 24: attacker:(UserA[],TicServA[],Nak[sid_2603])
Rule 25:
attacker:((UserA[]),v_2615,ShareEnc((TicServA[],v_2616,Nak[sid_2617],v_
2618),ltkAKAS[])) ->
attacker:(AppServX[],Nat[(UserA[]),v_2615,ShareEnc((TicServA[],v_2616,N
ak[sid_2617],v_2618),ltkAKAS[]),sid_2617],v_2615,ShareEnc((UserA[],Tat[
(UserA[]),v_2615,ShareEnc((TicServA[],v_2616,Nak[sid_2617],v_2618),ltkA
KAS[]),sid_2617]),v_2616))
Rule 26:
attacker:(UserA[],v_2629,ShareEnc((AppServX[],v_2630,Nat[(UserA[]),v_26
31,ShareEnc((TicServA[],SharedKey_2632,Nak[sid_2633],v_2634),ltkAKAS[])
,sid_2633],v_2635),SharedKey_2632)) &
attacker:((UserA[]),v_2631,ShareEnc((TicServA[],SharedKey_2632,Nak[sid_
2633],v_2634),ltkAKAS[])) ->
attacker:(v_2629,ShareEnc((UserA[],Tab[v_2629,ShareEnc((AppServX[],v_26
30,Nat[(UserA[]),v_2631,ShareEnc((TicServA[],SharedKey_2632,Nak[sid_263
3],v_2634),ltkAKAS[]),sid_2633],v_2635),SharedKey_2632),(UserA[]),v_263
1,ShareEnc((TicServA[],SharedKey_2632,Nak[sid_2633],v_2634),ltkAKAS[]),
sid_2633]),v_2630))
Rule 27:
attacker:ShareEnc((Tab[v_2640,ShareEnc((AppServX[],SharedKey_2641,Nat[(
UserA[]),v_2642,ShareEnc((TicServA[],SharedKey_2643,Nak[sid_2644],v_264
5),ltkAKAS[]),sid_2644],v_2646),SharedKey_2643),(UserA[]),v_2642,ShareE
nc((TicServA[],SharedKey_2643,Nak[sid_2644],v_2645),ltkAKAS[]),sid_2644
]),SharedKey_2641) &
attacker:(UserA[],v_2640,ShareEnc((AppServX[],SharedKey_2641,Nat[(UserA
[]),v_2642,ShareEnc((TicServA[],SharedKey_2643,Nak[sid_2644],v_2645),lt
kAKAS[]),sid_2644],v_2646),SharedKey_2643)) &
attacker:((UserA[]),v_2642,ShareEnc((TicServA[],SharedKey_2643,Nak[sid_
2644],v_2645),ltkAKAS[])) ->
attacker:ShareEnc(AttackSuccess[],SharedKey_2641)
Rule 28: attacker:(v_2652,v_2653,v_2654) ->
attacker:(v_2652,ShareEnc((v_2652,Kxt[(v_2652,v_2653,v_2654),sid_2655],
Tk[(v_2652,v_2653,v_2654),sid_2655]),ltkKASTGS[]),ShareEnc((v_2653,Kxt[
(v_2652,v_2653,v_2654),sid_2655],v_2654,Tk[(v_2652,v_2653,v_2654),sid_2
655]),ltkAKAS[]))
```

```
Rule 29:
attacker:(v_2671,v_2672,ShareEnc((v_2673,SharedKey_2674,v_2675),ltkKAST
GS[]),ShareEnc((v_2673,v_2676),SharedKey_2674)) ->
attacker:(v_2673,ShareEnc((v_2673,Kxaps[v_2671,v_2672,ShareEnc((v_2673,
SharedKey_2674,v_2675),ltkKASTGS[]),ShareEnc((v_2673,v_2676),SharedKey_
2674),sid_2677],Tt[v_2671,v_2672,ShareEnc((v_2673,SharedKey_2674,v_2675
),ltkKASTGS[]),ShareEnc((v_2673,v_2676),SharedKey_2674),sid_2677]),ltkA
PSTGS[]),ShareEnc((v_2671,Kxaps[v_2671,v_2672,ShareEnc((v_2673,SharedKe
y_2674,v_2675),ltkKASTGS[]),ShareEnc((v_2673,v_2676),SharedKey_2674),si
d_2677],v_2672,Tt[v_2671,v_2672,ShareEnc((v_2673,SharedKey_2674,v_2675)
,ltkKASTGS[]),ShareEnc((v_2673,v_2676),SharedKey_2674),sid_2677]),Share
dKey_2674))
Rule 30:
attacker:(ShareEnc((v_2691,SharedKey_2692,v_2693),ltkAPSTGS[]),ShareEnc
((v_2691,v_2694),SharedKey_2692)) ->
attacker:ShareEnc(v_2694,SharedKey_2692)
Completing...
ok, secrecy assumption verified: fact unreachable attacker:ltkAKAS[]
ok, secrecy assumption verified: fact unreachable attacker:ltkKASTGS[]
ok, secrecy assumption verified: fact unreachable attacker:ltkAPSTGS[]
Starting query not attacker:AttackSuccess[]
RESULT not attacker:AttackSuccess[] is true.
```

Listing 26: ProVerif Kerberos Observed Correctness Test Data

2. Performance Criteria

a. Execution Time

Table 35 presents the execution time data collected for the full version of the NS Protocol 2 as discussed in [NEED1978] for ProVerif. From this the CI^{+} value for the execution time was calculated; the Execution Time CI^{+} value is 0.09 sec.

Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)
0	0.09	10	0.09	20	0.09	30	0.09	40	0.09
1	0.09	11	0.09	21	0.09	31	0.09	41	0.09
2	0.09	12	0.09	22	0.09	32	0.09	42	0.09
3	0.08	13	0.09	23	0.09	33	0.08	43	0.09
4	0.09	14	0.09	24	0.09	34	0.09	44	0.09
5	0.09	15	0.09	25	0.09	35	0.09	45	0.09
6	0.09	16	0.09	26	0.09	36	0.09	46	0.09
7	0.09	17	0.09	27	0.09	37	0.09	47	0.09
8	0.09	18	0.09	28	0.09	38	0.09	48	0.09
9	0.09	19	0.09	29	0.09	39	0.09	49	0.09
-	-	-	-	-	-	-	-	50	0.09

Table 35: ProVerif NS Full 2 Execution Time Criterion Test Data

Table 36 presents the execution time data collected for the full version of the NSL Protocol 2 as discussed in [LOWE1996] for ProVerif. From this the CI^{+} value for the execution time was calculated; the Execution Time CI^{+} value is 0.05 sec.

Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)
0	0.05	10	0.05	20	0.05	30	0.05	40	0.05
1	0.05	11	0.05	21	0.05	31	0.05	41	0.05
2	0.05	12	0.05	22	0.05	32	0.05	42	0.05
3	0.05	13	0.05	23	0.05	33	0.05	43	0.05
4	0.05	14	0.05	24	0.05	34	0.05	44	0.05
5	0.05	15	0.05	25	0.05	35	0.05	45	0.05
6	0.05	16	0.05	26	0.05	36	0.05	46	0.05
7	0.05	17	0.05	27	0.05	37	0.05	47	0.05
8	0.05	18	0.05	28	0.05	38	0.05	48	0.05
9	0.05	19	0.05	29	0.05	39	0.05	49	0.05
-	-	-	-	-	-	-	-	50	0.05

Table 36: ProVerif NSL Full 2 Execution Time Criterion Test Data

Table 37 presents the execution time data collected for the Kerberos protocol as discussed in [RFC4120] for ProVerif. From this the CI^{+} value for the execution time was calculated; the Execution Time CI^{+} value is 0.03 sec.

Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)	Trial	Time (sec)
0	0.03	10	0.03	20	0.03	30	0.03	40	0.03
1	0.03	11	0.03	21	0.03	31	0.03	41	0.03
2	0.03	12	0.03	22	0.03	32	0.03	42	0.03
3	0.03	13	0.03	23	0.03	33	0.03	43	0.03
4	0.03	14	0.03	24	0.03	34	0.03	44	0.03
5	0.03	15	0.03	25	0.03	35	0.03	45	0.03
6	0.03	16	0.03	26	0.03	36	0.03	46	0.03
7	0.03	17	0.03	27	0.03	37	0.03	47	0.03
8	0.03	18	0.03	28	0.03	38	0.03	48	0.03
9	0.04	19	0.03	29	0.03	39	0.03	49	0.03
-	-	-	-	-	-	-	-	50	0.03

Table 37: ProVerif Kerberos Execution Time Criterion Test Data

b. Secondary Memory Requirement

Listing 27 presents the secondary memory requirement data collected for ProVerif. The total secondary memory required for ProVerif is 6.124 MB.

```
/usr/bin/proverif1.13pl7:
total 1316
/usr/bin/proverif1.13pl7/docs:
total 448
/usr/bin/proverif1.13pl7/examples:
total 64
/usr/bin/proverif1.13pl7/examples/auth:
total 80
/usr/bin/proverif1.13pl7/examples/choice:
total 196
/usr/bin/proverif1.13pl7/examples/ffgg:
total 16
/usr/bin/proverif1.13pl7/examples/jfk:
total 64
```

```
/usr/bin/proverif1.13pl7/examples/piboth:
total 268
/usr/bin/proverif1.13pl7/examples/pinoninterf:
total 160
/usr/bin/proverif1.13pl7/examples/secr:
total 508
/usr/bin/proverif1.13pl7/examples/weaksecr:
total 124
/usr/bin/proverif1.13pl7/src:
total 2880
/usr/bin/proverif1.13pl7/tests:
total 0
```

Listing 27: ProVerif Secondary Memory Requirement Criterion Test Data

c. Main Memory Requirement

Table 38 presents the main memory requirement data collected for the full version of the NS Protocol 2 as discussed in [NEED1978] for ProVerif. From this the CI^+ value for the main memory requirement was calculated; the Main Memory CI^+ value is 0 KB.

Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)
0	0	10	0	20	0	30	0	40	0
1	0	11	0	21	0	31	0	41	0
2	0	12	0	22	0	32	0	42	0
3	0	13	0	23	0	33	0	43	0
4	0	14	0	24	0	34	0	44	0
5	0	15	0	25	0	35	0	45	0
6	0	16	0	26	0	36	0	46	0
7	0	17	0	27	0	37	0	47	0
8	0	18	0	28	0	38	0	48	0
9	0	19	0	29	0	39	0	49	0
-	-	-	-	-	-	-	-	50	0

Table 38: ProVerif NS Full 2 Main Memory Requirement Criterion Test Data

Table 39 presents the main memory requirement data collected for the full version of the NSL Protocol 2 as discussed in [LOWE1996] for ProVerif. From this the CI^+ value for the main memory requirement was calculated; the Main Memory CI^+ value is 0 KB.

Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)
0	0	10	0	20	0	30	0	40	0
1	0	11	0	21	0	31	0	41	0
2	0	12	0	22	0	32	0	42	0
3	0	13	0	23	0	33	0	43	0
4	0	14	0	24	0	34	0	44	0
5	0	15	0	25	0	35	0	45	0
6	0	16	0	26	0	36	0	46	0
7	0	17	0	27	0	37	0	47	0
8	0	18	0	28	0	38	0	48	0
9	0	19	0	29	0	39	0	49	0
-	-	-	-	-	-	-	-	50	0

Table 39: ProVerif NSL Full 2 Main Memory Requirement Criterion Test Data

Table 40 presents the main memory requirement data collected for the Kerberos protocol as discussed in [RFC4120] for ProVerif. From this the CI^+ value for the main memory requirement was calculated; the Main Memory CI^+ value is 0 KB.

Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)	Trial	MMR (KB)
0	0	10	0	20	0	30	0	40	0
1	0	11	0	21	0	31	0	41	0
2	0	12	0	22	0	32	0	42	0
3	0	13	0	23	0	33	0	43	0
4	0	14	0	24	0	34	0	44	0
5	0	15	0	25	0	35	0	45	0
6	0	16	0	26	0	36	0	46	0
7	0	17	0	27	0	37	0	47	0
8	0	18	0	28	0	38	0	48	0
9	0	19	0	29	0	39	0	49	0
-	-	-	-	-	-	-	-	50	0

Table 40: ProVerif Kerberos Main Memory Requirement Criterion Test Data

APPENDIX D: LIST OF PROTOCOL ANALYSIS SYSTEMS

Here we present a list of protocol analysis systems. This list is by no means exhaustive.

Short Name	Long Name	Website
AVISPA	Automated Verification of Internet Security Protocols and Applications	http://www.avispa-project.org
CAPSL	Common Authentication Protocol Specification Language	http://www.csl.sri.com/users/millen/capsl/
CPSA	Cryptographic Protocol Shapes Analyzer	No public website.
Isabelle	Isabelle	http://www.cl.cam.ac.uk/Research/HVG/Isabelle/
Maude	Maude	http://maude.cs.uiuc.edu/
μCRL	Micro CRL	http://homepages.cwi.nl/~mcrl
Mocha	Mocha	http://embedded.eecs.berkeley.edu/research/mocha
MuCAPSL	Multicast Common Authentication Protocol Specification Language	http://www.csl.sri.com/users/millen/capsl/muintro.html
Murφ	Murφ	http://verify.stanford.edu/dill/murphi.html

Short Name	Long Name	Website
PDA	Protocol Derivation Assistant	http://www.kestreltechnology.com/soultions/pda.php
PRISM	Probabilistic Symbolic Model Checker	http://www.cs.bham.ac.uk/~dxp/prism/index.php
ProVerif	ProVerif	http://www.di.ens.fr/~blanchet/crypto-eng.html
PVS	Prototype Verification System	http://pvs.csl.sri.com/

Table 41: List of Protocol Analysis Systems

LIST OF REFERENCES

1. Abadi, M., and Blanchet, B., "Analyzing Security Protocols with Secrecy Types and Logic Programs," *Journal of the ACM*, vol. 52, no. 1, pp. 102-146, Jan. 2005 [ABAD2005].
2. Abadi, M., and Blanchet, B., "Computer-Assisted Verification of a Protocol for Certified Email," *10th International Static Analysis Symposium (SAS '03)*, San Diego, CA, pp. 316-335, Jun. 2003 [ABAD2003].
3. Abadi, M., Blanchet, B., and Fournet, C., "Just Fast Keying in the Pi Calculus," *13th European Symposium on Programming (ESOP '04)*, Barcelona, Spain, pp. 340-354, Mar. 2004 [ABAD2004].
4. Basin, D., Mödersheim, S., and Viganó, L., "An On-the-Fly Model-Checker for Security Protocol Analysis," Swiss Federal Institute of Technology Zurich, Zurich, Switzerland, Technical Report 404, Apr. 2003 [BASI2003].
5. Blanchet, B., "An Efficient Cryptographic Protocol Verifier Based on Prolog Rules," *14th IEEE Computer Security Foundations Workshop (CSFW-14)*, Cape Breton, Nova Scotia, Canada, pp. 82-96, Jun. 2001 [BLAN2001].
6. Burrows, M., Abadi, M., and Needham, R., "A Logic of Authenticaion," *ACM Transactions on Computer* Systems, vol. 8, no. 1, pp. 18-36, Feb. 1990 [BURR1990].
7. Clark, J., and Jacob, J., "A Survey of Authentication Protocol Literature: Version 1.0," *University of York*, Nov. 1997 [CLAR1997].
8. Committee on National Security Systems Instruction 4009 *National Information Assurance (IA) Glossary*, Government Printing Office, Washington, D.C., May 2003 [CNSS4009].
9. De Veaux, R. D., Velleman, P. F., and Bock, D. E., *Stats: Data and Models*, San Francisco: Pearson Addison Weasley, 2005 [DEVE2005].

10. Doghmi, S., Herzog, J., Guttman, J., and Thayer, J., "Automating Strand Spaces Protocol Analysis," *Protocol Exchange Workshop*, Jun. 2005 [DOGH2005].
11. Guttman, J., Herzog, J., Doghmi, S., Thayer, J., and Segall, A., "Cryptographic Shapes Analyzer (CPSA) Manual," The MITRE Corporation, Bedford, MA, Sep. 2006 [GUTT2006].
12. Healy, K., Coffey, T., and Dojen, R., "A Comparative Analysis of State-Space Tools for Security Protocol Verification," *WSEAS Transactions on Information Science and Applications*, vol. 1, no. 5, pp. 1256-1261, Nov. 2004 [HEAL2004].
13. Herzog, J., "Short Tutorial: CPSA," *Protocol Exchange Workshop*, Sep. 2005 [HERZ2005].
14. Holzman, G., "Basic Spin Manual," AT&T Bell Laboratories, Murray Hill, NJ, Technical Report, 1994 [HOLZ1994].
15. Hopper, N., Seshia, S., and Wing, J., "Combining Theory Generation and Model Checking for Security Protocol Analysis," Carnegie Mellon University, Pittsburg, PA, Technical Report CMU-CS-00-107, Jan. 2000 [HOPP2000].
16. Hjort, H., Hananel, D., and Lucas, D., "Quality Functional Deployment and Integrated Product Development," *Journal of Engineering Design*, vol. 3, no. 1, pp. 17-29, 1992 [HJOR1992].
17. IETF RFC 4120, The Kerberos Authentication Service (V5), Jul. 2005 [RFC4120].
18. Libes, D., *Exploring Expect: A Tcl-Based Toolkit for Automating Interactive Programs*, Sebastopol: O'Reilly, 1995 [LIBE1995].
19. López, H., "Frameworks for the Analysis of Security Protocols, A Survey," Pontificia Universidad Javeriana – Cali, Cali, Columbia, Research Report, Sep. 2006 [LOPE2006].

20. Lowe, G., “Breaking and Fixing the Needham-Schroeder Public Key Protocol Using FDR,” *Proceedings of TACAS*, vol. 1055, pp. 147-166, 1996 [LOWE1996].
21. Lowe, G., “Casper: A Compiler for the Analysis of Security Protocols,” *10th IEEE Computer Security Foundations Workshop*, Rockport, MA, pp. 18-30, Jun. 1997 [LOWE1997].
22. Meadows, C., “Formal Methods for Cryptographic Protocol Analysis: Emerging Issues and Trends,” *Journal on Selected Areas of Communication*, vol. 21, no. 1, pp. 44-54, Jan. 2003 [MEAD2003].
23. Needham, R., and Schroeder, M., “Using Encryption for Authentication in Large Networks of Computers,” *Communications of the ACM*, vol. 21, no. 12, Dec. 1978 [NEED1978].
24. Nielsen, J., *Usability Engineering*, San Francisco: Morgan Kaufman, 1993, pp. 165-226 [NIEL1993].
25. Saul, E., and Hutchinson, A., “SPEAR II – The Security Protocol Engineering and Analysis Resource,” *Second South African Telecommunications, Networks and Applications Conference*, 1999 [SAUL1999].
26. Saul, E., “Facilitating the Modeling and Automated Analysis of Cryptographic Protocols,” Master’s Thesis, University of Cape Town, Jul. 2001 [SAUL2001b].
27. Saul, E., and Hutchinson, A., “Enhanced Security Protocol Engineering Through a Unified Multidimensional Framework,” *Journal on Selected Areas of Communications*, vol. 21, no. 1, pp. 62-76, Jan. 2003 [SAUL2003].
28. Thayer, J., Herzog, J., and Guttman, J., “Strand Spaces: Proving Security Protocols Correct,” *Journal of Computer Security*, vol. 7, no. 2-3, pp. 191-230, Jan. 1999 [THAY1999].

THIS PAGE INTENTIONALLY LEFT BLANK

FURTHER READING

1. Gong, L., "Network Authentication Protocols: Lower Bounds and Optimal Implementations," *Distributed Computing*, vol. 9, no. 3, pp. 131-145, 1995.

THIS PAGE INTENTIONALLY LEFT BLANK

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Fort Belvoir, Virginia

2. Dudley Know Library
   Naval Postgraduate School
   Monterey, California

3. Professor YEO Tat Soon
   Director, Temasek Defence Systems Institute
   National University of Singapore
   Singapore

4. TAN Lai Poh
   Temasek Defence Systems Institute
   National University of Singapore
   Singapore